1. Abstract

Methods, systems, and software for installing and operating selected software applications on a client computer that is in communication with a server computer on a computer network are described. In one aspect of the present invention, a method for controlling the degree of access to operating system resources for a software program running on a computer that is running said operating system is provided. The degree of access to the operating system resources is defined for the software program, and at least one file including instructions for executing the software program is loaded on the computer from the server computer. The file is examined to determine the degree of system-level access available to the software program when the software program is being executed by the computer. The software program is executed, and a program instruction associated with the software program is intercepted when the software is being executed on the computer. A determination is then made to determine if the program instruction includes an operation that is outside of a degree of system-level access that is available to the software program, and if it is determined that the software program has permission to access system-level resources associated with the computer that are within the degree of system-level access available to the software, the program instruction is executed.

2. Representative Drawing

Fig. 4

(19)日本国特許庁（ＪＰ）　(12) **公開特許公報**（Ａ）　(11)特許出願公開番号

特開平10－254783

(43)公開日　平成10年(1998)９月25日

(51)Int.Cl.[6]	識別記号	ＦＩ	
Ｇ０６Ｆ　12/14	３１０	Ｇ０６Ｆ　12/14	３１０Ｋ
9/445		9/06	４２０Ｊ

審査請求　未請求　請求項の数19　ＯＬ　外国語出願　（全 63 頁）

(21)出願番号　特願平10－35321

(22)出願日　平成10年(1998)１月９日

(31)優先権主張番号　０８／７８０８２３

(32)優先日　1997年１月９日

(33)優先権主張国　米国（ＵＳ）

(71)出願人　595034134
サン・マイクロシステムズ・インコーポレイテッド
Ｓｕｎ　Ｍｉｃｒｏｓｙｓｔｅｍｓ，　Ｉｎｃ．
アメリカ合衆国　カリフォルニア州　94303　パロ　アルト　サン　アントニオ　ロード　901

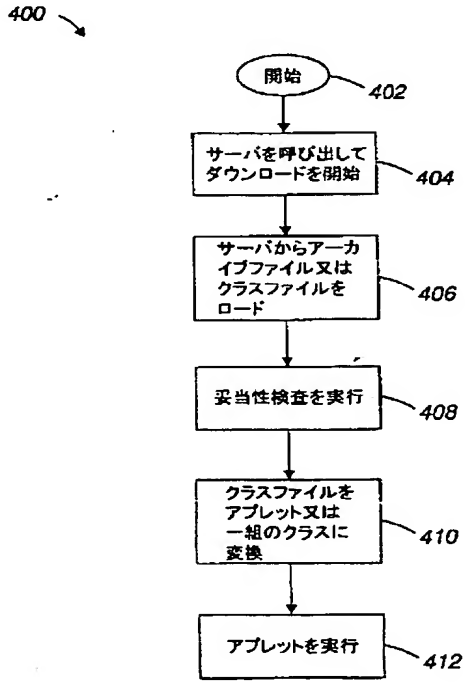
(72)発明者　ナタラジュ　ナガラトナム
アメリカ合衆国，　ニュー　ヨーク州，　シラキューズ，　ユークリッド　アヴェニュー　620，　アパートメント　1

(74)代理人　弁理士　長谷川　芳樹　（外５名）

最終頁に続く

(54)【発明の名称】　システムリソースへのソフトウェアアクセスを制御する方法及び装置

(57)【要約】
【課題】　コンピュータリソースへのアクセスを制御する好適な技術を提供する。
【解決手段】　オペレーティングシステムリソースへのアクセス度がソフトウェアプログラムに対して定義され、ソフトウェアプログラムを実行する命令を含むファイルがコンピュータにロードされる。このファイルを調べることにより、ソフトウェアプログラムの実行中にソフトウェアプログラムが利用できるシステムレベルアクセス度が求まる。ソフトウェアプログラムが実行されると、ソフトウェアプログラムに関連付けられたプログラム命令はソフトウェアの実行中に中断される。ソフトウェアプログラムが利用できるシステムレベルアクセス度の範囲外のオペレーションをプログラム命令が含んでいるか判断され、利用可能なシステムレベルアクセス度内のシステムレベルリソースにアクセスする許可をソフトウェアが有していると判断された場合、プログラム命令が実行される。



【特許請求の範囲】
【請求項１】　オペレーティングシステムを走らせているコンピュータ上で走るソフトウェアプログラムのためのオペレーティングシステムリソースへのアクセスの程度を制御する方法であって、
（ａ）前記ソフトウェアプログラムのための前記オペレーティングシステムリソースへのアクセスの前記程度を定義するステップと、
（ｂ）前記ソフトウェアプログラムに関連付けられた少なくとも一つのファイルを調べ、前記ソフトウェアプログラムが前記コンピュータにより実行されているときに前記ソフトウェアプログラムが利用できるシステムレベルアクセスの程度を判定するステップと、
（ｃ）前記コンピュータ上で前記ソフトウェアプログラムを実行するステップと、
（ｄ）前記ソフトウェアプログラムが前記コンピュータ上で実行されているときに前記ソフトウェアプログラムに伴うプログラム命令をインターセプトするステップと、
（ｅ）前記ソフトウェアプログラムが利用できるシステムレベルアクセスの前記程度を外れるオペーレーションを前記プログラム命令が含んでいるかどうかを判断するステップと、（ｆ）前記ソフトウェアプログラムが利用できるシステムレベルアクセスの程
度内の前記コンピュータに関連付けられたシステムレベルリソースにアクセスする許可を前記ソフトウェアプログラムが有していると判断された場合に、前記プログラム命令を実行するステップと、を備える方法。
【請求項２】　前記ソフトウェアプログラムが利用できるシステムレベルアクセスの前記程度を外れるオペーレーションを前記プログラム命令が含んでいるかどうかを判断する前記ステップは、前記ソフトウェアプログラムに関連付けられた識別子の妥当性を検査するステップを含んでいる、請求項１記載の方法。
【請求項３】　前記プログラム命令を実行する前記ステップは、前記プログラム命令によってアクセスされている前記システムレベルリソースが保護されたシステムレベルリソースであるかどうか判断するステップを含んでいる、請求項１又は２記載の方法。
【請求項４】　前記ソフトウェアプログラムがアプレットを含んでいる請求項１～３のいずれか記載の方法。
【請求項５】　前記アプレットは、Ｊａｖａアプレットである、請求項４記載の方法。
【請求項６】　前記アプレットはヘッダに関連付けられており、このヘッダーは識別子を含むように構成されており、この識別子は、前記ファイルの起源を識別するように構成されている、請求項４又は５記載の方法。
【請求項７】　前記識別子の妥当性を検査して、前記コンピュータが前記システムレベルリソースにアクセスする許可を有しているかどうかを判断するステップを更に備える請求項６記載の方法。
【請求項８】　前記コンピュータは、クライアントコンピュータであり、前記アプレットは、サーバコンピュータから前記クライアントコンピュータにダウンロードされる、請求項４～７のいずれか記載の方法。
【請求項９】　（ａ）前記調べるステップは、前記アプレットが前記クライアントコンピュータにより実行されているときに前記アプレットが利用できる前記サーバへのシステムレベルアクセスの程度を、前記アプレットのための前記サーバコンピュータに関連付けられた前記システムレベルリソースへのアクセスの程度を定義する前記ステップによって定義されたように判定するステップを含んでおり、
（ｂ）前記判断ステップは、前記サーバコンピュータに関連付けられたシステムレベルリソースにアクセスする前記プログラム命令が、前記アプレットが利用できるシステムレベルアクセスの前記程度を外れるオペレーションを含むかどうかを判断するステップを含んでおり、
（ｃ）前記実行ステップは、前記アプレットが利用できるシステムレベルアクセスの程度内の前記サーバコンピュータに関連付けられたシステムレベルリソースにアクセスする許可を前記アプレットが有していると判断された場合に、前記サーバコンピュータに関連付けられたシステムレベルリソースにアクセスする前記プログラム命令を実行するステップを含んでいる、請求項８記載の方法。
【請求項１０】　オペレーティングシステムを走らせているクライアントコンピュータ上で走るソフトウェアプログラムのためのオペレーティングシステムリソースへのアクセスの程度を制御する方法であって、前記オペレーティングシステムリソースの少なくとも一部が、コンピュータネットワークを介して前記クライアントコンピュータに結合されているサーバコンピュータ上に存在し、
（ａ）前記ソフトウェアプログラムのための前記オペレーティングシステムリソースへのアクセスの前記程度を定義するステップと、
（ｂ）前記ソフトウェアプログラムを前記クライアントコンピュータ上で実行する命令を含む少なくとも一つのファイルをロードするステップと、
（ｃ）前記少なくとも一つのファイルを調べ、前記ソフトウェアプログラムが前記クライアントコンピュータにより実行されているときに前記ソフトウェアプログラムが利用できるシステムレベルアクセスの程度を、前記アクセスの程度を定義する前記ステップによって定義されたように判定するステップと、
（ｄ）前記ソフトウェアプログラムを前記クライアントコンピュータ上で実行するステップと、
（ｅ）前記ソフトウェアプログラムが前記クライアントコンピュータ上で実行されているときに、前記ソフトウ

ェアプログラムに関連付けられたプログラム命令をインターセプトするステップと、
（ｆ）前記ソフトウェアプログラムが利用できるシステムレベルアクセスの前記程度を外れるオペーレーションを前記プログラム命令が含んでいるかどうかを判断するステップと、
（ｇ）前記ソフトウェアプログラムが利用できるシステムレベルアクセスの程度内のシステムレベルリソースにアクセスする許可を前記ソフトウェアプログラムが有していると判断された場合に前記プログラム命令を実行するステップと、
を備える方法。
【請求項１１】　前記ソフトウェアプログラムが利用できるシステムレベルアクセスの前記程度を外れるオペーレーションを前記プログラム命令が含んでいるかどうか判断する前記ステップは、前記ソフトウェアプログラムに関連付けられた識別子の妥当性を検査するステップを含んでいる、請求項１０記載の方法。
【請求項１２】　前記プログラム命令を実行する前記ステップは、前記プログラム命令によってアクセスされている前記システムレベルリソースが保護されたシステムレベルリソースであるかどうかを判断するステップを含んでいる、請求項１０又は１１記載の方法。
【請求項１３】　前記コンピュータネットワークを介して前記クライアントコンピュータと前記サーバコンピュータとの間にデータ転送通信リンクを確立するステップと、前記コンピュータネットワークを介して前記サーバコンピュータから前記クライアントコンピュータに前記少なくとも一つのファイルを伝送するステップと、を更に備える請求項１０～１２記載の方法。
【請求項１４】　クライアントからの要求を処理して、第１のサーバに伴うシステムリソースにアクセスする方法であって、
（ａ）第２のサーバを呼び出して、前記要求に関連するファイルのダウンロードを開始するステップと、
（ｂ）前記第２サーバから前記関連ファイルをロードするステップであって、前記関連ファイルは、アーカイブファイルを含み、前記アーカイブファイルは、少なくとも一つのクラスファイルと、ヘッダと、を含み、前記ヘッダは、前記アーカイブファイルの起源を表示するように構成された識別子を含んでいるステップと、
（ｃ）前記アーカイブファイルの妥当性を検査するステップと、
（ｄ）前記クラスファイルをアプレットに変換するステップと、
（ｅ）前記アプレットを実行するステップであって、前記アプレットは少なくとも一つの命令を含んでおり、前記アプレットを実行することにより、前記クライアントが、前記第１サーバに伴う前記システムリソースにアクセスすることが可能となるステップと、を備える方法。
【請求項１５】　前記アーカイブファイルの妥当性を検査する前記ステップは、
（ａ）前記ヘッダを認証するサブステップと、
（ｂ）前記ヘッダが妥当であるかどうかを判断するサブステップと、
（ｃ）前記ヘッダが妥当であると判断された場合に、前記クラスに対してクラス確認を行うサブステップと、
を含んでいる請求項１４記載の要求処理方法。
【請求項１６】　前記アプレットを実行する前記ステップは、
（ａ）前記命令が保護されたオペレーションを実行する命令であるかどうかを判断するサブステップと、
（ｂ）前記命令が保護されたオペレーションを実行する命令ではないと判断された場合に、前記オペレーションを実行するサブステップと、
（ｃ）前記命令が保護されたオペレーションを実行する命令であると判断された場合に、前記オペレーションが許されているかどうかを判断するサブステップと、を含んでいる請求項１４又は１５記載の要求処理方法。
【請求項１７】　オペレーティングシステムリソースへのアクセスの程度を制御するコンピュータシステムであって、
ソフトウェアプログラムを実行する命令を含む少なくとも一つのファイルを内部に保持する少なくとも一つの記憶装置に結合された第１のコンピュータを備えており、
前記ソフトウェアプログラムは前記第１コンピュータ上で走り、前記第１コンピュータは前記オペレーティングシステムを走らせ、
前記第１コンピュータは、
（ａ）前記ソフトウェアプログラムのための前記オペレーティングシステムリソースへのアクセスの前記程度を定義し、
（ｂ）前記第１コンピュータ上で前記ソフトウェアプログラムを実行する命令を含んだ前記少なくとも一つのファイルをロードし、
（ｃ）前記少なくとも一つのファイルを調べて、前記ソフトウェアプログラムが前記第１コンピュータにより実行されているときに前記ソフトウェアプログラムが利用できるシステムレベルアクセスの程度を判定し、
（ｄ）前記ソフトウェアプログラムを前記第１コンピュータ上で実行し、
（ｅ）前記ソフトウェアプログラムが前記第１コンピュータ上で実行されているときに、前記ソフトウェアプログラムに関連付けられたプログラム命令をインターセプトし、
（ｆ）前記ソフトウェアプログラムが利用できるシステムレベルアクセスの前記程度を外れるオペレーションを前記プログラム命令が含むかどうかを判断し、
（ｇ）前記ソフトウェアプログラムが利用できるシステムレベルアクセスの程度内の前記第１コンピュータに関

連付けられたシステムレベルリソースにアクセスする許可を前記ソフトウエアプログラムが有していると判断された場合に前記プログラム命令を実行するように構成されている、コンピュータシステム。

【請求項１８】　前記第１コンピュータは、前記プログラム命令によりアクセスされている前記システムレベルリソースが保護されたシステムレベルリソースであるかどうかを判断するように構成されている、請求項１７記載のコンピュータシステム。

【請求項１９】　コンピュータ読取り可能プログラムコード装置を備えるコンピュータ読取り可能媒体であって、前記コンピュータ読取り可能プログラムコード装置は、

（ａ）前記ソフトウェアプログラム用の前記オペレーティングシステムリソースへのアクセスの前記程度を定義し、

（ｂ）前記ソフトウェアプログラムに関連付けられた少なくとも一つのファイルを調べて、前記ソフトウェアプログラムがコンピュータにより実行されているときに前記ソフトウェアプログラムが利用できるシステムレベルアクセスの程度を判定し、

（ｃ）前記ソフトウェアプログラムを前記コンピュータ上で実行し、

（ｄ）前記ソフトウェアプログラムが前記コンピュータ上で実行されているときに、前記ソフトウェアプログラムに関連付けられたプログラム命令をインターセプトし、

（ｅ）前記ソフトウェアプログラムが利用できるシステムレベルアクセスの前記程度を外れるオペレーションを前記プログラム命令が含むかどうかを判断し、

（ｆ）前記ソフトウェアプログラムが利用できるシステムレベルアクセスの程度内の前記コンピュータに関連付けられたシステムレベルリソースにアクセスする許可を前記ソフトウェアプログラムが有していると判断された場合に前記プログラム命令を実行するという動作を前記コンピュータに行わせるように構成されている、コンピュータ読取り可能媒体。

【発明の詳細な説明】

【０００１】

【発明の属する技術分野】本発明は、全般的に、コンピュータ上で走るソフトウェアによりコンピュータリソースへのアクセスを制御する方法および装置に関する。特に、本発明は、サーバコンピュータからクライアントコンピュータにダウンロードされたソフトウェアによりクライアントコンピュータ上のシステムリソースへのアクセスを制御する方法および装置に関する。

【０００２】

【従来の技術】パーソナルコンピュータが現れる前、コンピュータユーザができることは、データやコマンドを入力するためのキーボードと出力を見るためのビデオ表示装置（またはプリンタ）を通常含む端末を用いて、大型のメインフレームコンピュータ上で走るソフトウェアを操作することに限られていた。メインフレームは、非常に強力なコンピューティングプラットフォームを提供したが、重大な欠点を抱えていた。具体的には、メインフレームは、設置と運用に費用がかかり、全てのユーザは端末を介してメインフレームに直結されることが必要とされ、これによって多くの人はメインフレームへのアクセスが制限されていた。更に、ユーザは、自らのコンピューティング環境において極めて限られた制御しか行えず、通常は、自らの作業スタイルや問題をメインフレームコンピュータのソフトウェアや管理に合わせなければならなかった。

【０００３】１９７０年代後半から、パーソナルコンピュータは、個人用、ビジネス用、及び科学利用のための有力なコンピューティングプラットフォームとしてメインフレームに追いつき始めた。パーソナルコンピュータは、多数の処理ジョブを同時に引き受けなければならなかった過去のメインフレームと同じ演算速度を単独のユーザに与えることが可能な場合も多かった。また、パーソナルコンピュータ上で走るソフトウェアは、いっそう「ユーザフレンドリ」になり、これによってコンピュータユーザは、自らの特定の計算ニーズにコンピュータとソフトウェアの双方を適合させることができようになった。端末をメインフレームに接続する必要から解放されるため、パーソナルコンピュータは、企業や家庭内の任意の場所に置くことができる。この能力により、必要な場所に置くことができるコンピューティングパワーとしてのパーソナルコンピュータのメインフレームに対する優位性は更に確実なものとなった。大型で高価で気難しいメインフレームコンピューティングセンターにユーザが自らの操作を合わせる必要は、もはやなくなったのである。

【０００４】１９８０年代を通じてパーソナルコンピュータの演算能力とデータ記憶容量が爆発的に増大したので、パーソナルコンピュータの優位性は確実になったように思われた。しかし、１９８０年代の終わりになって、過去２０年のパーソナルコンピュータ革命に追いつく可能性のある新しい現象が現れ始めた。今日では、依然として増え続けるパーソナルコンピュータが、高速のデータネットワークを介して相互にリンクされている。現在、最も普及しているネットワークは、「インターネット」である。インターネットは、地球上の様々な商用コンピュータサイトや学術コンピュータサイトや個人コンピュータサイトから構成されるネットワークである。このインターネットの人気、特に「ワールドワイドウェブ（WorldWide Web）」と呼ばれるインターネットの一面の人気が、「イントラネット」と呼ばれることの多い内部的なコンピュータネットワークの形成を多くの企業に促すこととなった。徐々に高性能になるネットワーク

サーバやルータや多数の独立コンピュータの効率の良い通信を可能にする他の装置と高速データネットワークとの組み合わせにより、ネットワークコンピューティングに対する関心に火がついたのである。

【0005】ワールドワイドウェブの魅力の一部は、その極めて視覚的な性格に由来する。これは、パーソナルコンピュータの出現と大型コンピュータに対するその優位性に大きな役割を果たした要因と同じである。通常、ワールドワイドウェブは様々な「ウェブサイト」に編成されており、このウェブサイトは、通常、「ブラウザ」を実行するクライアントコンピュータにデータを転送するサーバを含んでいる。ブラウザは、ウィンドウと種々のコントロールをユーザに提供するソフトウェアであり、このブラウザを介してサーバからのデータを見たりナビゲートしたりすることができる。ワールドワイドウェブデータの特に有用な特徴は、ユーザがマウスボタン操作のような非常に簡単で直感的なコマンドを通じて一方のドキュメントから他方のドキュメントへ、さらに一方のウェブサイトから他方のウェブサイトへさえも素早く進むことができるようにデータがハイパーテキストコマンドを通じてリンクされるというその能力にある。ワールドワイドウェブを利用することにより、ユーザは、世界中のサイトからテキストやグラフィックを見たりダウンロードしたりサウンドを聴いたりすることができる。更に、ユーザは、新しいソフトウェアや、クライアントコンピュータに既にインストールされているプログラムを修正できるソフトウェアをダウンロードすることもできる。

【0006】インターネット上のワールドワイドウェブのユーザが利用できるこれらの特徴は、ローカルネットワークのユーザにも「イントラネット」を介して同じように提供することができる。ここでイントラネットとは、インターネットと同様に構成されるクライアント及びサーバを含んだ非公開のコンピュータネットワークである。従業員の任務遂行に有用な情報をネットワークを通じて企業内のパーソナルコンピュータに速やかに配布できるので、この能力に対して多くの企業からの関心が高まってきている。特に、多くの企業は、イントラネットを用いて、そのようなイントラネットを利用する企業内の個人にデータベースやカスタムソフトウェアプログラムへのアクセスを提供している。例えば、Ｊａｖａ（商標）プログラミング言語（カリフォルニア州パロアルトのSun Microsystemsが市販）を用いて作られるカスタムソフトウェアアプレットは、イントラネットの内部または外部のコンピュータに既に組み込まれたソフトウェア及びデータと共同して操作することができ、これにより、従来から行われていた特殊目的ソフトウェアの多数のコピーの配布や保守に付随する困難を伴うことなくユーザが自らのジョブタスク専用のデータ及びソフトウェアにアクセスできるようにすることができる。

【0007】安全なイントラネットを経由して配布されるソフトウェアは、クライアントコンピュータのシステムリソースへ完全にアクセスできることが望ましい場合が多いが、イントラネットシステムの外部にある安全性の低いネットワークを介して配布されるソフトウェアは必ずしも信頼できないので、このようなソフトウェアには、ファイル移動機能のようなシステムリソースへのアクセスはほとんど又は全く許されない。例えば、ソフトウェアアプリケーションのなかには、コンピュータウイルスをホストコンピュータにインストールする機能を備えるものがある。また、ホストコンピュータからの重大なデータをコピー、変更、あるいは消去するソフトウエアアプリケーションや、そのデータを他のコンピュータシステムへ不正に転送することさえ行うようなソフトウェアアプリケーションもあるかもしれない。信頼できるソフトウェアには特定リソースへのアクセスを許可する一方で他のソフトウェアに対しては同じリソースへのアクセスを制限する実現可能な方法や装置は、残念ながら存在しない。従って、全てのソフトウェア（信頼できるものも怪しいものも）に対して全てのシステムリソースへのアクセスを許可することと、クライアントシステムのセキュリティを保護するために全てのソフトウェアのアクセスを制限することとの間のトレードオフがユーザに残されることになる。

【0008】

【発明が解決しようとする課題】このように、高度分散形コンピュータネットワークに伴う上記の問題を軽減できるようにネットワーク上の情報とソフトウェアの双方に対するリソースアクセスを制御する方法やシステムを提供することは、コンピュータユーザ、特に多数のコンピューユーザがコンピュータネットワークを介して結ばれている企業のコンピューユーザ、にとって大変有益である。以下で述べるように、本発明は、これらのニーズやその他のニーズを満足する。

【0009】

【課題を解決するための手段】本発明は、ネットワーク化されたコンピュータを介したソフトウェア配布を管理する際の上記の問題に取り組むものであり、この取り組みは、ある態様では、第１のコンピュータ上で選択されたソフトウェアアプリケーションによるサーバリソースへのアクセスを制御する方法、システム、およびソフトウェアを提供することによって行われる。ここで、この第１コンピュータは、コンピュータネットワーク上でサーバコンピュータとして機能する第２のコンピュータと通信を行うクライアントコンピュータとして機能するものである。

【0010】本発明の一態様では、コンピュータ上で走るソフトウェアプログラムのためのオペレーティングシステムリソースへのアクセスの程度を制御する方法が提供される。オペレーティングシステムリソースへのアク

セス度がソフトウェアプログラムに対して定義され、このソフトウェアプログラムを実行する命令を含む少なくとも一つのファイルがコンピュータにロードされる。このファイルは、ソフトウェアプログラムがコンピュータによって実行されているときにソフトウェアプログラムが利用できるシステムレベルアクセスの程度を判定するために調べられる。ソフトウェアプログラムが実行されると、ソフトウェアプログラムに関連付けられた安全なリソースへのアクセスを要求するプログラム命令は、ソフトウェアがコンピュータ上で実行されているときにインターセプトされる。次いで、ソフトウェアプログラムが利用できるシステムレベルアクセスの程度を外れるオペレーションをプログラム命令が含むかどうかの判断がなされ、ソフトウェアが利用できるシステムレベルアクセスの程度内にあるシステムレベルリソースであってコンピュータに関連付けられたシステムレベルリソースにアクセスする許可をソフトウェアプログラムが有していると判断された場合、プログラム命令が実行される。

【００１１】本発明の別の態様では、オペレーティングシステムリソースのうちの少なくとも一部が、クライアントコンピュータに結合されたサーバコンピュータ上に存在している場合に、オペレーティングシステムを走らせているクライアントコンピュータ上で走るソフトウェアプログラムのためのシステムリソースへのアクセスの程度を制御する方法が提供される。ソフトウェアプログラム用のオペレーティングシステムリソースへのアクセス度が定義され、クライアントコンピュータ上でソフトウェアプログラムを実行する命令を含んだ少なくとも一つのファイルがロードされる。このファイルは、クライアントコンピュータによるこのソフトウェアプログラムの実行中にソフトウェアプログラムが利用できるシステムレベルアクセスの程度を判定するために調べられる。このソフトウェアプログラムがクライアントコンピュータ上で実行されると、このソフトウェアプログラムに伴うプログラム命令は、ソフトウェアプログラムがクライアントコンピュータ上で実行されているときにインターセプトされる。ソフトウェアプログラムが利用できるシステムレベルアクセスの程度を外れる演算をプログラム命令が含むかどうかに関する判断がなされ、ソフトウェアプログラムが利用できるシステムレベルアクセスの程度内にあるシステムレベルリソースにアクセスする許可をソフトウェアプログラムが有していると判断された場合、プログラム命令が実行される。

【００１２】上記した態様や利点、ならびに本発明の他の態様や利点は、添付の図面とともに以下の記載を読むと明らかになる。

【００１３】

【発明の実施の形態】本発明、及び本発明の更なる利点は、添付図面をとともになされる以下の説明を参照することにより最もよく理解することができる。

【００１４】アプレットによるシステムリソースへのアクセスを制御する方法と装置の実施形態を、添付図面を参照しながら以下で説明する。

【００１５】本発明による一つのネットワークが、図１に示されている。図１で説明されているネットワークには、イントラネット１０２および１０４、ならびに符号１０６で示される個人の外部コンピュータが含まれている。イントラネット１０２および１０４の構造は、図２を参照しながら以下で更に詳細に説明する。イントラネットおよびユーザの双方は、様々なコンピュータゲートウェイ（“G／W”）を介してコンピュータネットワークに接続されている。一部の実施形態では、このコンピュータネットワークにインターネットが含まれる。図１に関して更に詳しく述べると、イントラネット１０２は、符号１０８で概略図示されているインターネットを介して、イントラネット１０４およびユーザ１０６に結合されている。イントラネット１０２とインターネット１０８との間の接続は、まずゲートウェイ１１０を介して行われる。ゲートウェイ１１０は、イントラネット１０２と、「バックボーン（backbone）」、すなわち高容量データライン１１２と、に結合されている。高容量ライン１１２からのデータは、ゲートウェイ１１４を介して伝送され、インターネット１０８を通り、第２ゲートウェイ１１６を通過して、符号１１８で示される高容量データラインに入る。コンピュータネットワーク技術の当業者であれば明らかなように、データライン１１８は、データライン１１２と同じものであってもよく、あるいは他の様々な個人、すなわちユーザとネットワークとが結合される別個のバックボーンを表していてもよい。

【００１６】イントラネット１０２からインターネット１０８を通り、さらに高速データライン１１８を越えて移動するデータは、ゲートウェイ１２０を通過してイントラネット１０４に至るか、またはゲートウェイ１２４を通過してユーザ１０６に至る。このように、ここで説明した実施形態によれば、ユーザ１０６、イントラネット１０４、およびイントラネット１０２の間でデータを受け渡すことができる。特に、データは、いま説明したようにインターネット１０８を通過して移動することができ、あるいはユーザ１０６とイントラネット１０４との間のバックボーン１１８を通過することができる。一部の実施形態では、イントラネット１０４とイントラネット１０２は、当業者に「エクストラネット(extranet)」として知られているネットワーク構成を介して直接結合させることができる。エクストラネットは、所定のネットワークや個人が専用データ接続を介してリモートネットワークに結合されるネットワーク構成である。この接続は、図１で説明されるようなインターネットを通じて送られるデータを含んでいてもよいが、直接的なデータ供給、例えばＩＳＤＮやＴ－１データラインを介す

るもの、であってもよい。種々の構成、およびこのような構成を確立する方法や材料は、コンピュータネットワークおよび電気通信技術の当業者には明らかである。

【００１７】図１において符号１０２または１０４で示されるようなイントラネットの一実施形態が、図２において符号５０で示されている。典型的なイントラネット５０は、クライアント６２および６４に結合されたサーバ６０を含んでいる。更に、サーバ６０は、ノード６８で示されるようなルータ、ハブまたは同様のデータ転送装置を介して、符号７０、７２、および７４で示されるような他のクライアントコンピュータに結合することができる。更に、リモートクライアント（図示せず）は、ダイレクトラインを通じて、またはモデムや同様の装置などを用いて電話回線を通じて、サーバ６０に接続することができる。一部の例では、イントラネット５０へのアクセスが、囲み７５によって示される「ファイアウォール」構成によって高度に制御される。ファイアウォールの外側にいるユーザ、例えばリモートクライアント７８、からの通信は、保護サーバへのアクセスを許可するゲートウェイを通過することにより確立することができる。このようなゲートウェイは符号７６で示されている。

【００１８】通常、サーバは、サーバと通信を行う種々のクライアントがアクセスすることのできるデータやソフトウェアを、直接に、またはルータなどの装置を介して、提供する。サーバ、ルータ、および種々のクライアントマシンの構造、メインテナンス、および動作は、当業者には周知である。幾つかの特定の実施形態では、サーバ６０は、ワールドワイドウェブ上のデータを見るために用いられるようなブラウザソフトウェアと互換性のあるデータを提供するように構成される。詳しくいえば、サーバ６０により提供されるデータは、代表的なブラウザソフトウェアを用いて調べることのできるページ形式のデータとなる。ある実施形態では、サーバおよびクライアントは、データのみならず、カリフォルニア州パロアルトのSun Microsystemsから市販されているＪａｖａ（商標）プログラミング言語で書かれた「アプレット」形式のコンピュータソフトウェアを交換するように構成されている。本明細書中で用いられる「アプレット」とは、ソースコンピュータ（一般的にはサーバ）からクライアントマシンに受け渡され、クライアントに既にインストールされているソフトウェアと共同して走るように構成されたソフトウェアプログラムのことである。ある実施形態では、アプレットとともに走るソフトウェアは、上記のブラウザソフトウェアである。通常、アプレットは、ブラウザソフトウェア自身が実行するように構成されていない種々の計算タスクを遂行することにより、ブラウザソフトウェアに追加機能を与える。このように、アプレットをダウンロードするユーザは、アプレットなしではブラウザソフトウェアが利用できない追加機能をブラウザソフトウェアに与えることができる。このような追加機能には、例えば、データベースに対するカスタムインタフェースが含まれていてもよい。

【００１９】一般に、クライアント、例えばクライアント６２は、ユーザエージェント、つまりブラウザ、を用いてサーバ６０への呼出しを行う。ユーザエージェントには、カリフォルニア州パロアルトのSun Microsystems社が市販するＨｏｔＪａｖａ（商標）や、カリフォルニア州マウンテンビューのNetscape Communications社が市販するＮｅｔｓｃａｐｅが含まれる。ただし、これらに限定されるものではない。ある実施形態では、ユーザエージェントは、一般に、アプレットコードを実行する処理エンジンと、アプレットが特定のシステムリソースへのアクセスを有するかどうかを判断する際に用いられるセキュリティマネージャと、を含んでいる。このようなセキュリティマネージャの例については後述する。

【００２０】ある実施形態では、インターネット上またはイントラネット内に配置されるサーバが、アプレットを定義するクラスファイルを含んだクラスライブラリを提供する。このようなクラスライブラリの一例が、Ｊａｖａ（商標）クラスライブラリである。具体的に述べると、サーバは、アプレットを構成するクラスファイルと、アプレットを参照するＨＴＭＬコードを含んだ特定のウェブページと、を含むことができる。

【００２１】本発明のある実施形態によれば、アプレットは、ソースコンピュータ、すなわちサーバからクライアントマシンにダウンロードされたクラスファイルからインスタンス生成することができる。これらのクラスファイルは、一つのアーカイブファイルにグループ化することができる。更に、アーカイブファイルは、ディジタル署名を付けるか、さもなければマークを付すことができるので、アーカイブファイルから生成されたアプレットの起源（origin）を確実に判断することができる。この後、アプレットが実行されているマシンがどのシステムリソースにアクセスできるかを判断するために、アーカイブファイルの署名を確認することができる。署名（シグネチャ）を用いることにより、アプレットによるクライアントマシンのシステムリソースへのアクセスを、例えばアプレットの起源であるサーバのセキュリティ状態を参照することにより、制御することができる。例えば、各クライアント上のアプレットに対応する許可が異なる場合があるという事実から、一つのクライアント上で動作するアプレットは、第２のクライアント上で動作する同一のアプレットとは異なるアクセス特権を有する場合がある。従って、このリソースアクセス制御により、安全なマシン、例えばリソースを含むマシンと同じイントラネット内のマシン、に対応するアプレットが、安全でないマシン、例えばインターネット上のマシン、に対応するアプレットよりもリソースへのアクセス範囲が広くなるようにすることができる。

【００２２】図３（ａ）は、本発明の一実施形態に係るクラスファイルの集合の概略図である。このクラスデータファイル集合のフォーマットは、サーバ上で一般的に用いられているが、署名を受け入れるようには構成されていない。すなわち、各クラスファイルは、通常、サーバ上に存在するクラスを定義する。このフォーマットは、この集合が任意の数のクラス、例えばクラス“１”２０２、クラス“２”２０４、およびクラス“Ｎ”２０６、を含むようになっている。クラスは、そのクラスからその後に構成される任意のアプレットに固有のデータおよびメソッド、すなわちデータを処理するステートメントのシーケンス、を定義するソフトウェア構造として定義してもよい。言い換えると、先に述べたように、アプレットは、前もって定義されたクラスをインスタンス生成することにより構成することができる。単一のクラスを用いて多数のアプレットを構成することも可能である。

【００２３】アプレットの実行には、システムリソースへアクセスする要求、またはコマンド、を必要とするのが普通である。アプレットは多くの異なるシステムリソースにアクセスするための命令を含んでいてもよいが、セキュリティを考慮すると、アプレットは、現在の設計制約の下では、特定のシステムリソースの全てへのアクセスを許可されるか、または特定のシステムリソースのいずれにもアクセスが許可されないかのどちらかとなる。上述のように、システムリソースアクセスに対するこの「オール・オア・ナッシング」アプローチは、イントラネットシステム内で走る、例えば起源の分かっているアプレットが、例えば「信頼できる」のに対し、イントラネットシステムの外部で走る同等のアプレットは安全でないと考えられるという点で、望ましくないことが多い。イントラネットシステム内で走るアプレットおよび外部で走る同等のアプレットには、通常、システムリソースに対する同じアクセス特権が与えられるので、イントラネットシステムのセキュリティを維持するため、これらのアプレットには、一般に何らのアクセス特権も与えられない。

【００２４】リソースへのアクセスからアプレットを選択的に制御する能力により、イントラネットシステム内のユーザは、リソースへのアクセスを個々のアプレットに基づいて制限することができる。アプレットをインスタンス生成するために用いられるクラスファイルに「署名（signature）」、つまり識別子を含めることは、イントラネット機構がアプレットを選択的に制御できるようにする一つの方法である。クラスファイルの起源がどこであるかを判断できるようにクラスファイルに署名を付ける、すなわちマーク付けすることにより、イントラネットシステムは、クラスファイルからインスタンス生成されたアプレットに対応する適切なアクセス特権を判断できるようになる。更に、クラスファイルに署名を付けることにより、クラスファイルが不正に変更されているかどうかに関する判断を行うことも可能になる。一群のクラスファイルにディジタル署名を付けることを可能にするアーカイブファイル構造については、図３（ｂ）を参照しながら後述する。

【００２５】ディジタル署名を付けることの可能なアーカイブファイルを提供することにより、アーカイブファイルに付随するクラスファイルから構成されるアプレット（イントラネットシステムの内外のいずれかにあるアプレット）は、イントラネットシステム内の選択されたシステムリソースにアクセスできるようになる。アーカイブファイルのディジタル署名をチェックすることにより、与えられたアプレットが不正に変更されたかどうか、およびどのコンピュータがアプレットに署名を付けたか、を判断することが可能になる。このため、アプレットの起源が安全な、即ち信頼できるホストなのか、あるいは安全でないホストなのかに基づいてアクセス特権を割り当てることができる。更に、一部の実施形態では、アクセス特権の割当てにより、ユーザは、信頼できるホストと信頼できないホストを判断できるようになる。

【００２６】図３（ｂ）は、本発明の実施形態に係るアーカイブファイルデータフォーマットの概略図である。ここで説明する実施形態では、アーカイブフォーマットは、Ｊａｖａ（商標）アーカイブ（ＪＡＲ）フォーマットである。アーカイブ２１０、すなちアーカイブファイルは、ヘッダ署名２１２を含んでいる。このヘッダ署名２１２は、アーカイブ２１０の妥当性を確認してアーカイブ２１０が利用できるアクセスのレベルを判定するためにユーザエージェントによって通常用いられる署名である。一般に、ヘッダ署名２１２は、アーカイブのサイズに応じた情報（但し、これに限定されるものではない）を含む他の情報を含んだ一般ヘッダの一部とすることの可能なディジタル署名である。アーカイブ２１０は、任意の数の関連クラス、例えばクラス“１”２０２、クラス“２”２０４、およびクラス“Ｎ”２０６、を有しており、これらのクラスからアプレットおよび関連オブジェクトがインスタンス生成される。

【００２７】更に、アーカイブ２１０は、例えばデータブロック２１４のような関連データブロックを有していてもよい。データブロック２１４は、イメージやテキストやアーカイブ２１０の一部と考えられる任意のデータを含んでいてもよい。ある実施形態では、データブロック２１４は、アーカイブ２１０に付随するクラス２０２、２０４および２０６を記述するテキストストリングを含んでいてもよい。他の実施形態では、アーカイブ２１０にデータブロックが含まれない場合もある。

【００２８】次に、図４（ａ）を参照しながら、クライアント側ディレクトリ構造の実施形態を本発明に従って説明する。クライアントを介してリソースへのアクセス

を要求するユーザは、一般に、ユーザディレクトリ３０２とインタフェースする。ユーザディレクトリ３０２は、ブラウザ、つまりユーザエージェント、に関する情報を含む対応ブラウザディレクトリ３０４を有している。このブラウザは、例えば上述したＨｏｔＪａｖａ（商標）ブラウザのような任意の適切なブラウザとすることができる。ブラウザディレクトリ３０４は、ユーザによって行われる要求に適したプロパティファイル３０６を含んでいる。プロパティファイル３０６は、通常、ユーザプリファランス項目（user preference item）３０８を含んでいる。このユーザプリファランス項目３０８は、一般的に、ユーザによって提供されるブラウザ仕様である。これらの仕様には、ブラウザセットアップに関するデータやブラウザに関連する動作特性が含まれていてもよい。但し、これらに限定されるものではない。

【００２９】プロパティファイル３０６は、ユーザによって行われる特定の要求に関連した情報を更に含んでいる。例えば、このような情報には、イメージデータブロック３１０、構成ファイル名３１２、およびグループ仕様ファイル名３１４が含まれていてもよい。ある実施形態では、イメージデータブロック３１０は、要求に対応する任意のイメージを識別するデータファイル名、すなわちストリングを含んでいる。構成ファイル名３１２は、要求されたリソースを対応するセキュリティ記述子に容易にマッピングするために用いられる構成ファイルを識別するストリングである。構成ファイルの一例については、図４（ｂ）を参照しながら以下で説明する。グループ仕様（「スペック」）ファイル名３１４は、図５（ｃ）を参照しながら後述するように、グループ仕様ファイルを識別するストリングである。

【００３０】図４（ｂ）は、本発明の実施形態に係る構成ファイルの構造の概略図である。構成ファイル３５０は、図４（ａ）を参照して上述したような構成ファイル名３１２によって識別される構成ファイルの一例である。構成ファイル３５０は、サーバ上、すなわちクライアントがアクセスしたいサーバ上のリソース３５４を、対応するアクセスファイル名３５６に関連付けるテーブル３５２を含んでいる。すなわち、テーブル３５２は、リソース“列”３５４内のエントリをアクセスファイル名“列”３５６内の対応するエントリに関連付ける。リソース３５４は、一般に、種々のシステムリソース、例えばファイル、ホスト、ソケット番号、を識別する分類子（classifier）である。アクセスファイル名３５６は、対応するアクセスファイルを識別する。このアクセスファイルは、セキュリティ記述子と、アクセスファイルが関連付けられているシステムリソースへのアクセスの制御に関係する他の情報と、を含んでいる。アクセスファイルの構造は、図５（ｃ）を参照して以下で更に詳細に説明する。二つ以上のリソース３５４が同一のセキュリティ記述子を共有できるという事実により、アクセスファイル名３５６とアクセスファイルは、二つ以上のリソース３５４に関連付けることができる。

【００３１】次に、図５（ｃ）を参照しながらアクセスファイルの構造を本発明の実施形態に従って説明する。アクセスファイル３６０は、一般に、プリンシパル（principal）３６２を許可３６４に関連付けるテーブル３６１を含んでいる。プリンシパル３６２は、個々のホストであってもよいし、あるいは複数のホストからなるグループであってもよい。例えば、“java.com”は、プリンシパル３６２である個々のホスト、つまりサーバであってもよい。この他に、“java.com”と“sun.com”が、グループのプリンシパル３６２を形成してもよい。一部の実施形態では、プリンシパル３６２が、特定のアーカイブの署名者（signer）であってもよい。許可３６４は、セキュリティ記述子のグルーピング（grouping）を与える。すなわち、許可３６４は、許可３６４が関連付けられたプリンシパル３６２がアクセスするリソースを指定するセキュリティ記述子のグルーピングである。

【００３２】図５（ｄ）は、本発明の実施形態に係るグループ仕様（「スペック」）ファイルフォーマットの概略図である。上述したように、図４（ａ）のグループ仕様ファイル名３１４は、グループ仕様ファイル、例えばグループ仕様ファイル３７０、を識別する。グループ仕様ファイル３７０は、グループ名３７２を任意の数のメンバ３７４に関連付けるテーブル３７１を含んでいる。グループ名３７２は、本質的には、メンバ３７４のグループを識別するために用いることのできる識別子である。例を挙げると、一つのグループ名、例えばグループ“１”３７２ａは、任意の数のメンバ、例えばメンバ“１”３７４ａおよびメンバ“２”３７４ｂ、に関連付けることができる。また、一つのメンバ、例えばメンバ“１”３７４ａを二つ以上のグループ名３７２に関連付けることもできる。

【００３３】図６は、本発明の実施形態に従ってリソースへのアクセス要求を実行する方法を示すプロセスフローチャートである。このプロセスは４０２から開始し、ステップ４０４では、要求クライアント、例えば図２のクライアント７４から、サーバ、例えば図２のサーバ６０に対して呼出しが行われ、図３（ａ）を参照して上述した少なくとも一つのクラスファイル、あるいは図３（ｂ）を参照して上述したアーカイブファイルのいずれかのダウンロードが開始される。この要求は、ユーザエージェント、すなわち前述のＨｏｔＪａｖａ（商標）ブラウザやＮｅｔｓｃａｐｅＮａｖｉｇａｔｏｒブラウザなどのブラウザを介してなされるクライアント呼出しに応答してサーバ上で受信される。少なくとも一つのクラスファイル、またはアーカイブファイルのいずれかのダウンロードは、リソースへのアクセス要求に応答して開始されるので、このダウンロードの開始はアプレットを実行する呼出しである。ある好適な実施形態では、アー

カイブファイルはＪＡＲファイルである。

【００３４】ステップ４０６では、アーカイブファイルまたはクラスファイルのいずれかが、サーバから要求クライアントに付随する記憶装置にロードされる。一般に、アーカイブファイル内にクラスがない場合、例えばクラスにディジタル署名が付されていない場合、にはクラスファイルがロードされ、クラスにディジタル署名が付されている場合には、アーカイブファイルがロードされる。アーカイブファイルは、対応するクラスファイルを有している。このため、アーカイブファイルのロードは、クラスファイルのロードを伴う。クラスファイルが記憶装置にロードされた後、ステップ４０８では、ロードされたファイルに対して妥当性検査プロセスが実行される。この妥当性検査プロセスは、アーカイブファイルがロードされた場合、ロードされたアーカイブファイルに付随するヘッダ署名が妥当であるかどうかを確認するステップを含んでいるが、この妥当性検査プロセスについては図７を参照して後述する。

【００３５】妥当性検査プロセスの後、ステップ４１０では、クラスファイルがアプレットに変換される。すなわち、ロードされたクラスファイル（ＪＡＲファイルの一部である場合もあれば、そうでない場合もある）をインスタンス生成することにより、記憶装置内でアプレットが生成される。アプレットが生成されると、ステップ４１２でアプレットファイルが実行される。アプレットの実行に伴うステップについては、図８を参照して後述する。

【００３６】図７は、本発明の実施形態に従ってクラスファイルの妥当性検査を行うステップ、すなわち図６のステップ４０８を示すプロセスフローチャートである。このプロセスは、スッテプ５０２から始まり、ステップ５０４では、アーカイブファイルまたはクラスファイルのどちらがロードされたかに関する判断がなされる。クラスファイルがロードされていた場合、プロセスフローはステップ５０６へ進み、ここで標準クラス検査（standard class verification）が行われる。クラスファイルのなかにセキュリティを脅かすものが含まれているかどうかを確かめるため、標準クラス検査には、通常、ロードされた全てのファイル、そして全てのクラスのチェックが含まれる。一部の実施形態では、仮想計算機、例えばＪａｖａ（商標）仮想計算機、のセキュリティが脅かされる可能性があるかどうかを判断するためのチェックが行われる。標準クラス検査方法は、当業者には一般的によく知られている。標準クラス検査が実行されれば、クラスファイルの妥当性を検査する本プロセスはステップ５２０で完了する。

【００３７】ステップ５０４での判断が、アーカイブファイルがロードされたということであれば、ステップ５０８において、アーカイブファイルのヘッダの妥当性検査、すなわち認証が行われる。アーカイブファイルの妥当性検査は、一般に、ヘッダ署名に基づくアーカイブファイルの起源の識別を伴う。すなわち、ヘッダ署名の起源を証明し、これによりアーカイブファイルの起源を証明するためのチェックが行われる。この妥当性検査には、アーカイブファイルに付随するデータがそのままであるかどうかのチェックも含まれる。一部の実施形態では、アーカイブファイルにヘッダ署名が含まれていなくてもよい。例えば、イントラネット内のアーカイブファイルには署名を付さなくてもよい。ステップ５１０では、ヘッダが妥当であるかどうかについて判断がなされる。アーカイブの内容が署名に対応していないなど、ヘッダが妥当でない場合、ステップ５１４において、エラーフラグ等が出される。ある実施形態では、エラーフラグが出ると、例外がスロー（throw）される。別の実施形態では、エラーフラグが出ると、メッセージが要求クライアントに戻される。エラーフラグが出された後、クラスファイルの妥当性を検査する本プロセスはステップ５２０で終了する。

【００３８】ステップ５１０でヘッダが妥当であることが分かると、プロセスフローはステップ５１０からステップ５１２へ移り、ここでアーカイブファイルに付随するクラスのうち妥当性検査をすべきものが残っているかどうかが判断される。妥当性検査をすべきクラスがある場合、ステップ５１６において、標準クラス検査が行われる。ステップ５０６で既に述べたように、標準クラス検査には、与えられたクラスに仮想計算機のセキュリティを脅かす可能性のあるものが含まれているかどうかのチェックが含まれる。例えば、与えられたクラスに仮想計算機上のファイルやメモリを上書きしうるものが含まれている場合、仮想計算機のセキュリティが脅かされる可能性がある。与えられたクラスに対して標準クラス検査が完了した後、プロセス制御はステップ５１２に戻り、ここで妥当性を検査すべきクラスが他にもあるかどうかについて判断がなされる。プロセス制御は、妥当性を検査すべきクラスがもはや残っていないとステップ５１２で判断されるまでステップ５１２と５１６との間でループし、その判断がなされた時点で、クラスファイルの妥当性を検査する本プロセスは５２０で完了する。

【００３９】図８は、本発明の実施形態に係るアプレットの実行に伴うステップを示すプロセスフローチャートである。つまり、図６のステップ４１２について説明する。このプロセスは６０２から始まり、ステップ６０４では、オペレーションを実行する命令をアプレットが含んでいるかどうかについて判断がなされる。このオペレーションは、一般に、システムレベルリソースにアクセスするための呼出しであってもよい。オペレーションを実行する命令をアプレットが含んでいない場合、アプレットを実行する本プロセスは６１６で終了する。オペレーションを実行する命令をアプレットが含んでいる場合、プロセスフローはステップ６０６へ移り、実行すべ

きオペレーションが、保護されたオペレーション、例えばセキュアードオペレーション（secured operation）、であるかどうかが判断される。すなわち、そのオペレーションがアクセスの制御されているオペレーションであるかどうかについて判断がなされる。オペレーションが保護されていないと判断された場合、ステップ６０８でオペレーションが実行され、プロセスフローはステップ６０４に戻り、ここで他のオペレーションを実行する命令があるかどうかの判断が行われる。

【００４０】実行命令中のオペレーションが保護されているとステップ６０６で判断された場合、プロセスフローはステップ６１０へ移り、アプレットセキュリティマネージャが呼び出される。セキュリティマネージャの呼出しプロセスは、図９を参照して以下で更に詳細に説明する。アプレットセキュリティマネージャは、通常、与えられたアプレットがアクセスすることのできるオペレーションを制御する。ある実施形態では、アプレットセキュリティマネージャは、Ｊａｖａ（商標）アプレットセキュリティマネージャである。ステップ６１２では、オペレーションが許されるかどうかが判断される。言い換えると、ステップ６１２は、実行されるべきオペレーションにアプレットがアクセスするかどうかの判断である。オペレーションが許される場合、ステップ６０８でオペレーションが実行される。プロセス制御は、ステップ６０８からステップ６０４に戻り、ここで他のオペレーションを実行する命令があるかどうかの判断が行われる。

【００４１】ステップ６１２における判断が、演算が許されていないという判断である場合、エラー状態が発生するが、これはステップ６１４で例外をスロー（throw）することにより実現することができ、アプレットを実行する本プロセスは６１６で終了する。一部の実施形態では、例外をスローするステップは、throw関数の呼出しを伴う場合がある。他の実施形態では、例外をスローするステップは、要求クライアントに付随するユーザエージェントにより表示することができるエラーメッセージの伝送を伴っていてもよい。更に他の実施形態では、エラー処理（error handling）により、ユーザがアプレットによるオペレーションの実行を承認するかどうかを問う形式でユーザとの対話が行われるようにすることもできる。このような実施形態において、アクセスファイルは、ユーザから供給される応答が永続的に記録されるように更新することができる可能性がある。

【００４２】次に、図９を参照して、セキュリティマネージャを呼び出すプロセス、すなわち図８のステップ６１０を説明する。ユーザエージェントは、一般に、一つの対応セキュリティマネージャしか有していない。セキュリティマネージャを呼び出すプロセスは７０２から始まり、ステップ７０４では、アプレットによって呼び出されているオペレーションが識別される。このオペレーションは多数のオペレーションのうちの任意の一つとすることができるが、このオペレーションは、一般的には、読出し操作または書込み操作である。プロセスフローはステップ７０４からステップ７０６へ進み、ここでオペレーションに対応するリソースの名前が識別される。一部の実施形態では、リソースの名前は、セキュリティマネージャに対する呼出しに渡されるので、すぐに識別される。しかし、リソースの名前が呼出しに渡されない場合は、図４（ａ）を参照して前述したプロパティファイルを用いて対応リソースを識別することができる。

【００４３】対応リソースがステップ７０６で識別されると、そのリソースに対応するアクセスファイルの名前が、アプレットに関連する構成ファイルを用いて識別される。この構成ファイルについては、図４（ｂ）を参照して既に説明した。次いで、ステップ７１０では、アプレットに対応する許可がアクセスファイルから取得される。一部の実施形態では、適切なアクセスファイルが、記憶装置内の実アクセスファイルの表現である場合がある。図５（ｃ）を参照して上述したアクセスファイルは、個々のホストまたはグループを一組の許可に関連付ける。許可が取得された後、セキュリティマネージャに対する呼出しは７１２で完了する。

【００４４】図１０は、本発明に係る典型的なコンピュータシステムを示している。このコンピュータシステム８３０は、主記憶装置８３４（通常はリードオンリーメモリ、すなわちＲＯＭ）および主記憶装置８３６（通常はランダムアクセスメモリ、すなわちＲＡＭ）を含む記憶装置に結合された任意の数のプロセッサ８３２（中央処理装置、またはＣＰＵとも呼ばれる）を含んでいる。この技術では周知のように、ＲＯＭ８３４は、データと命令をＣＰＵに対して単方向に転送するように動作し、ＲＡＭ８３６は、通常、データと命令を双方向方式で転送するために用いられる。双方の主記憶装置８３４、８３６は、上述したような任意の適切なコンピュータ読取り可能媒体を含んでいてもよい。大容量記憶装置８３８も、ＣＰＵ８３２に双方向に結合されており、データ記憶容量を追加している。この大容量記憶装置８３８は、プログラム、データなどを記憶するために用いることができ、通常は、主記憶装置８３４、８３６よりも低速のハードディスクなどの二次記憶媒体である。大容量記憶装置８３８は、磁気テープ読取り装置や紙テープ読取り装置、あるいは他の周知装置の形をとっていてもよい。大容量記憶装置８３８内に保存される情報は、適切な場合には、仮想記憶装置としてＲＡＭ８３６の一部として標準方式で組み込むことができる。ＣＤ－ＲＯＭ８３４のような特殊な大容量記憶装置も、データをＣＰＵに単方向で転送することができる。

【００４５】ＣＰＵ８３２は、一つ以上の入出力装置８４０にも結合されている。この入出力装置には、ビデオ

モニタ、トラックボール、マウス、キーボード、マイクロフォン、タッチセンシティブディスプレイ、トランスジューサカード読取り装置、磁気テープ読取り装置、紙テープ読取り装置、タブレット、スタイラス、音声認識装置、手書き認識装置、あるいは周知の入力装置（もちろん、他のコンピュータもこの例である）などの装置が含まれる。但し、これらに限定されるわけではない。最後に、ＣＰＵ８３２は、符号８１２で全体的に示されるようなネットワーク接続を用いて、コンピュータネットワークや通信ネットワーク、例えばインターネットネットワークやイントラネットネットワーク、に結合されていてもよい。このようなネットワーク接続を用いることで、上述の方法ステップを実行している間、ＣＰＵは、ネットワークから情報を受け取ったり、情報をネットワークに出力することができると考えられる。上述の装置や材料は、コンピュータハードウェアおよびソフトウェア技術の当業者には良く知られているものである。更に、上記ハートウェアおよびソフトウェアの構成要素、ならびにネットワーク化装置が標準的な設計と構成を有していることは、当業者であれば理解することができる。

【００４６】本明細書で説明されているコンピュータ実施方法は、コンピュータプログラム命令をコンピュータシステム上で実行するコンピュータサイエンス技術において周知の技術と装置を用いて実施することができる。本明細書で用いられている用語「コンピュータシステム」は、データや命令を処理する処理装置（例えば中央処理装置、ＣＰＵ）を含んでいると定義されており、ここでこの処理装置は、処理装置との間でデータや命令を交換する一つの以上のデータ記憶装置、例えばＲＡＭ、ＲＯＭ、ＣＤ－ＲＯＭ、ハードディスク等（但し、これらに限定されない）、に結合されている。このデータ記憶装置は、処理装置専用、すなわち処理装置に直接結合されていてもよく、あるいはリモート、すなわちコンピュータネットワークを介して処理装置に結合されていてもよい。コンピュータネットワークを介して処理装置に結合されているリモートデータ記憶装置は、特定のワークステーション上での実行のためにプログラム命令を処理装置に送ることができる。更に述べると、上記の処理装置は、一つ以上の追加処理装置に同じ物理構造を通じて結合されていてもよいし（例えばパラレルプロセッサ）、あるいはコンピュータネットワークを介して結合されていてもよい（例えば分散形プロセッサ）。このようにリモート結合されたデータ記憶装置およびプロセッサの使用は、コンピュータサイエンス技術の当業者にはよく知られている（例えば、Ralston 1993を参照）。本明細書で用いられている用語「コンピュータネットワーク」は、相互に通信を行うことが可能な一組のコンピュータシステムと相互接続する一組の通信チャネルを含んでいると定義されている。これらの通信チャネルは、ツイストペア線、同軸ケーブル、光ファイバ、衛星リンク、デジタルマイクロ波無線などの伝送媒体を含んでいてもよい。但し、これらに限定されるものではない。コンピュータシステムは、大きな、すなわち「広い」エリアにわたって分散させたり（例えば、数十、数百、数千マイルにわたって分散するもの、ＷＡＮ）、あるいはローカルエリアネットワーク（例えば、数フィートから数百フィートにわたるもの、ＬＡＮ）とすることができる。更にまた、様々なローカルエリアネットワークや広域ネットワークを組み合わせて、複数のコンピュータシステムからなる集合ネットワークを形成することができる。このようなコンピュータネットワーク連合の一例が「インターネット」である。

【００４７】本発明の実施形態を数例しか説明しなかったが、本発明は、本発明の趣旨や範囲から逸脱することなく他の多くの具体的な形態で実施することができる。例えば、署名を付すことのできるアーカイブファイルデータ構造の構成は一つしか説明しなかったが、アーカイブファイルデータ構造は、本発明の範囲内で広範な変形を加えることが可能である。更に、システムリソースへのアクセス要求を実行する方法に伴うステップは、順序を並び替えることができる。また、本発明の趣旨や範囲から逸脱することなくステップを削除または追加することもできる。従って、ここで説明した実施形態は例示であって、これらに限定されるものではなく、本発明は、特許請求の範囲およびその均等物の全範囲によって定められる。

【図面の簡単な説明】

【図１】ユーザとイントラネットの双方がインターネットを介してコンピュータネットワークにより結合されている広域コンピュータネットワークの概略図である。

【図２】従来のイントラネットシステムの概略図である。

【図３】（ａ）は本発明の一実施形態に係るクラスファイルの集合の概略図であり、（ｂ）は本発明の一実施形態に係るアーカイブファイルデータフォーマットの概略図である。

【図４】（ａ）は本発明の一実施形態に係るクライアント側ディレクトリ構造の概略図であり、（ｂ）は本発明の一実施形態に係るクライアント側構成ファイルの構造の概略図である。

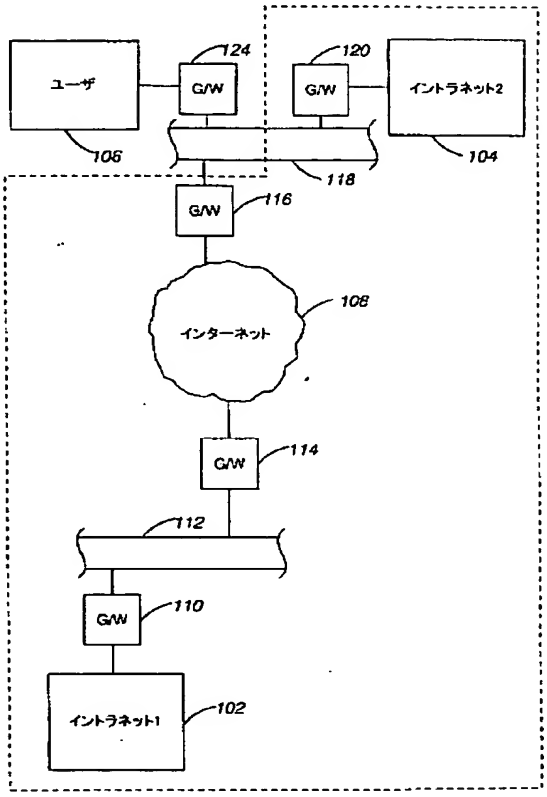
【図５】（ｃ）は本発明の一実施形態に係るクライアント側アクセスファイルの構造の概略図であり、（ｄ）は本発明の一実施形態に係るクライアント側グループ仕様ファイルの構造の概略図である。

【図６】本発明の一実施形態に従ってリソースへのアクセス要求を実行する方法を示すプロセスフローチャートである。

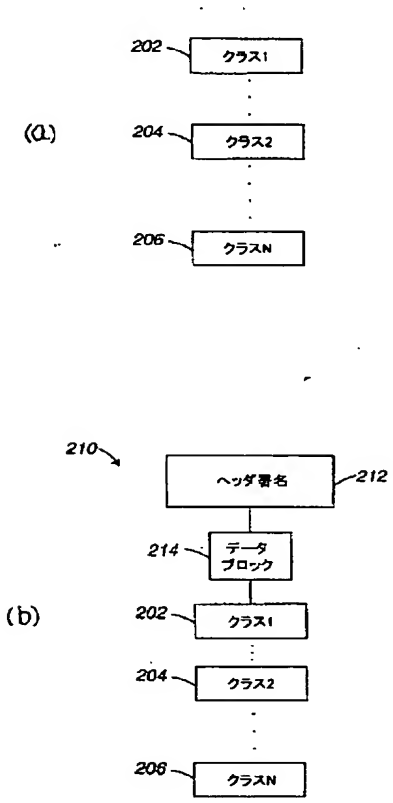
【図７】本発明の一実施形態に係るクラスファイルの妥当性検査に伴うステップを示すプロセスフローチャート

である。
【図８】本発明の一実施形態に係るアプレットの実行に伴うステップを示すプロセスフローチャートである。
【図９】本発明の一実施形態に係るセキュリティマネージャの呼出しに伴うステップを示すプロセスフローチャートである。
【図１０】本発明の一実施形態に係るコンピュータシステムの概略図である。
【符号の説明】
１…、２…、３…、。

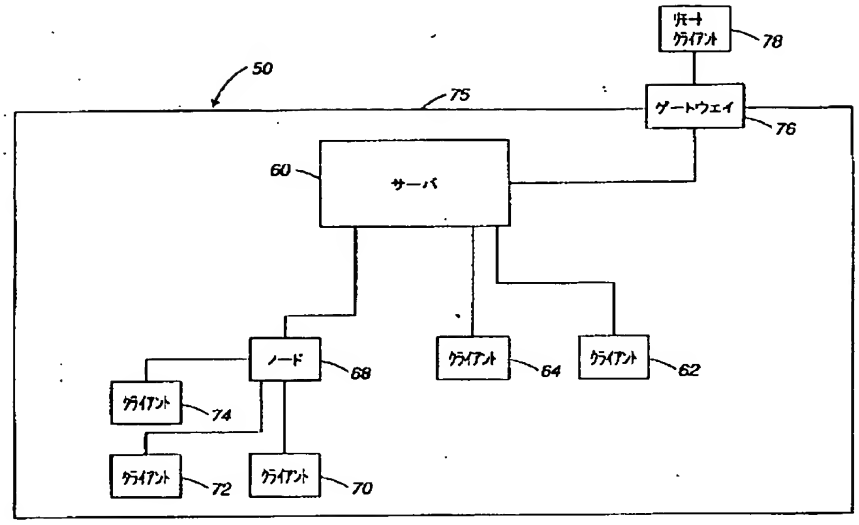
【図１】



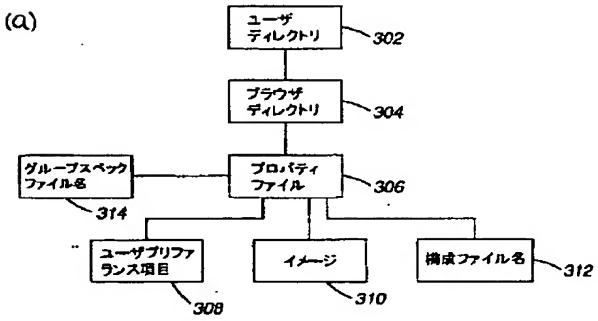
【図３】

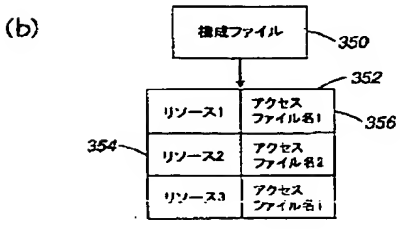


【図2】

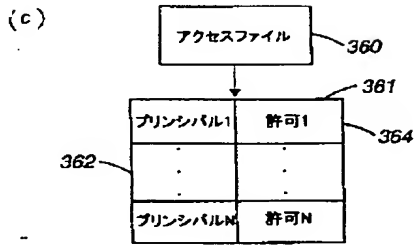

50
75
78
76
60
サーバ
ノード
68
64
62
74
72
70

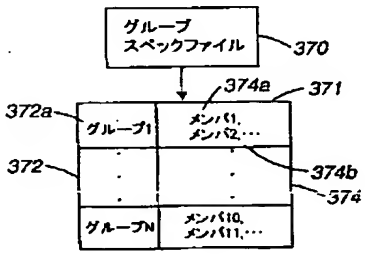
【図4】


(a)
ユーザ ディレクトリ
302
ブラウザ ディレクトリ
304
グループスペック ファイル名
314
プロパティ ファイル
306
ユーザプリファ ランス項目
308
イメージ
310
構成ファイル名
312

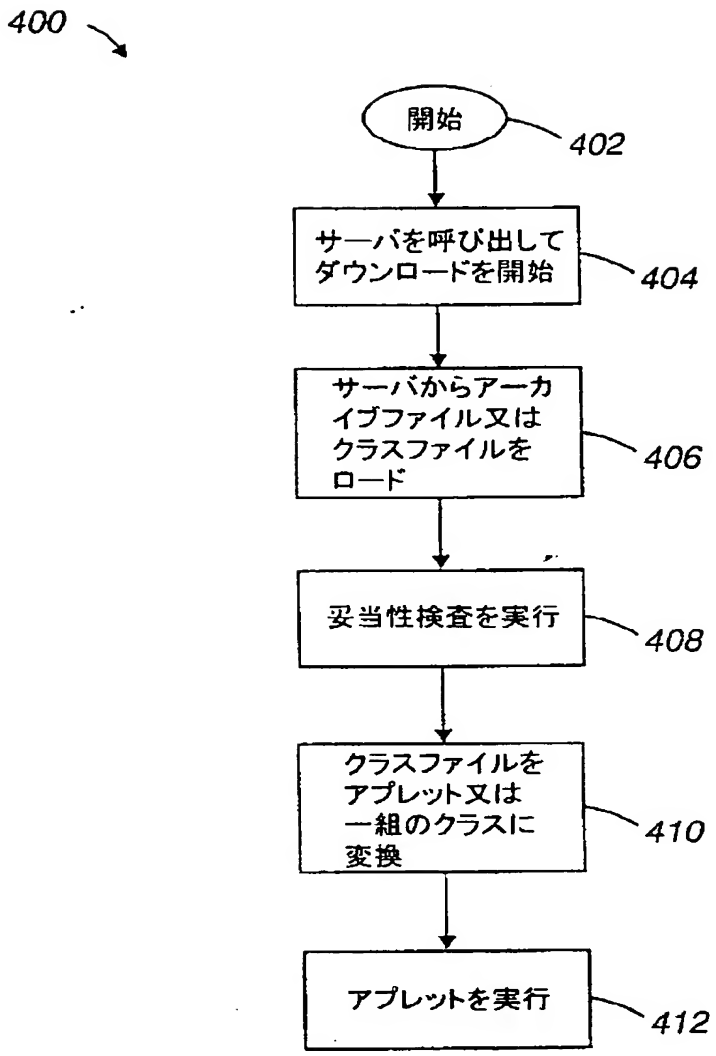

(b)
構成ファイル
350
352
リソース1
アクセス ファイル名1
356
354
リソース2
アクセス ファイル名2
リソース3
アクセス ファイル名3

【図5】

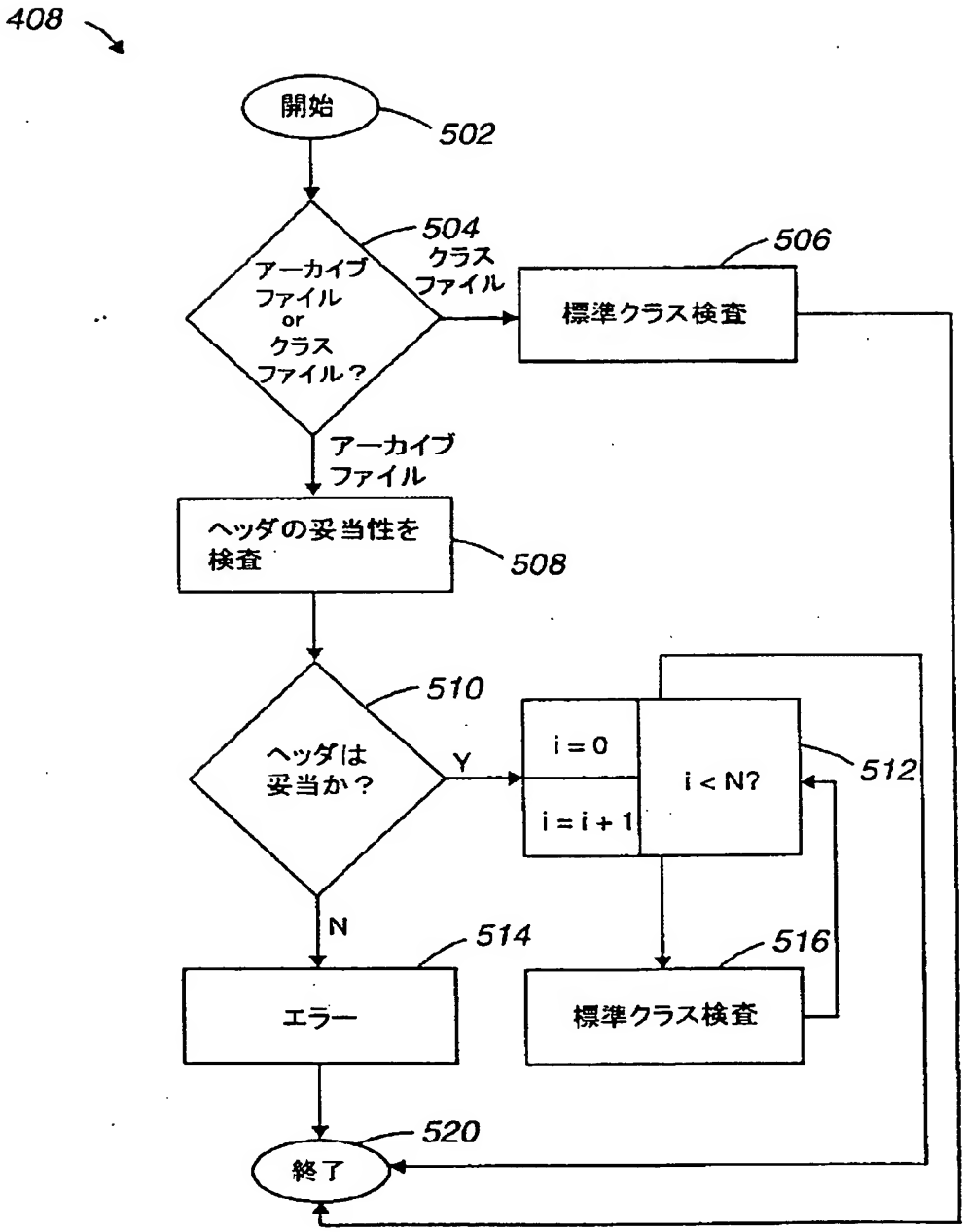

(c)
アクセスファイル
360
361
プリンシパル1
許可1
364
362
プリンシパルN
許可N


(d)
グループ スペックファイル
370
374a
371
372a
グループ1
メンバ1, メンバ2,…
372
374b
374
グループN
メンバ10, メンバ11,…

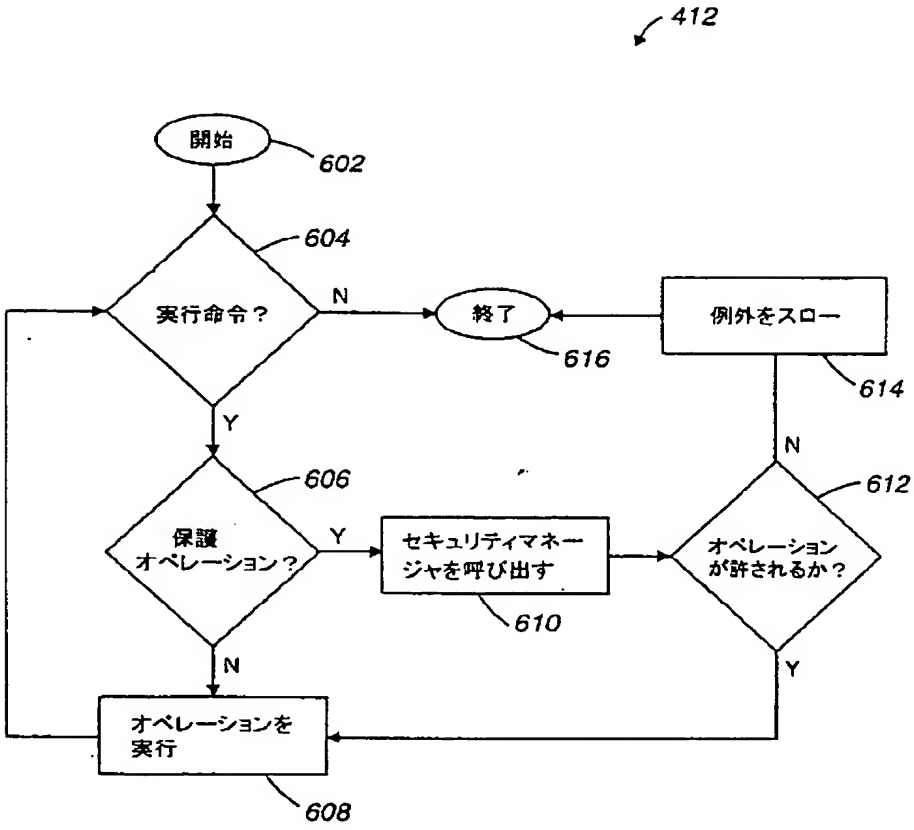
【図6】


400
開始
402
サーバを呼び出して
ダウンロードを開始
404
サーバからアーカ
イブファイル又は
クラスファイルを
ロード
406
妥当性検査を実行
408
クラスファイルを
アプレット又は
一組のクラスに
変換
410
アプレットを実行
412

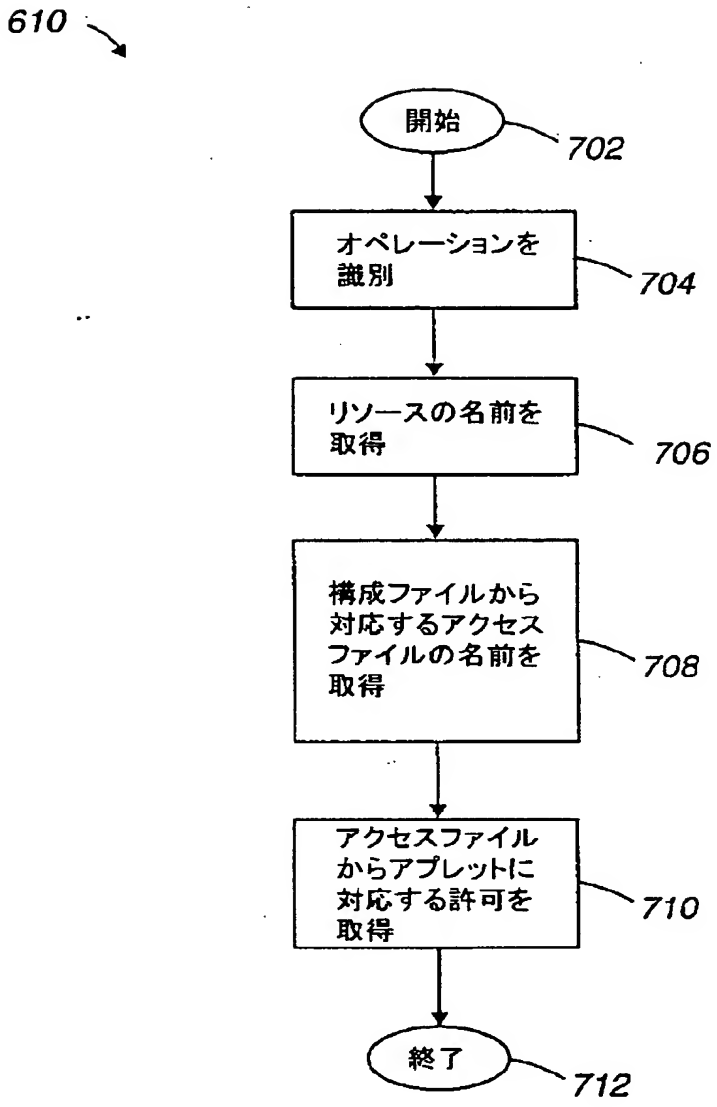
【図7】


408
開始
502
アーカイブ
ファイル
or
クラス
ファイル？
504
クラス
ファイル
506
標準クラス検査
アーカイブ
ファイル
ヘッダの妥当性を
検査
508
ヘッダは
妥当か？
510
Y
i＝0
i＝i＋1
i＜N?
512
N
エラー
514
標準クラス検査
516
終了
520

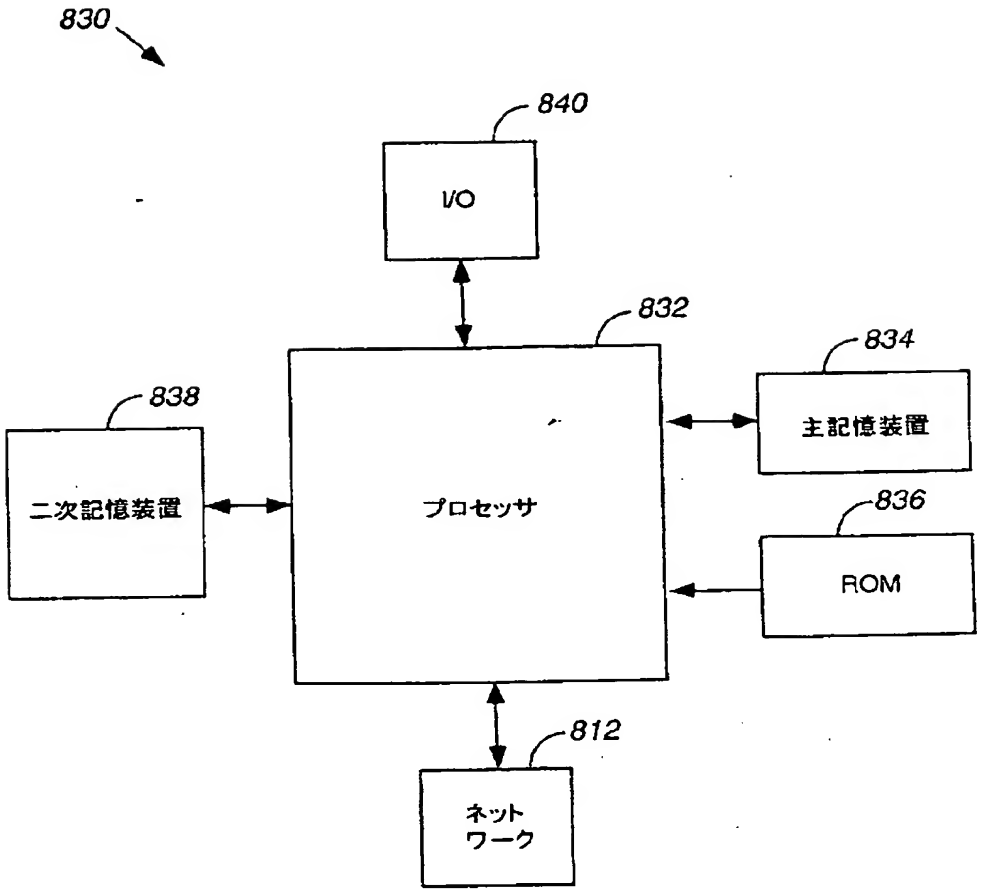
【図8】


412
開始
602
604
実行命令？
N
終了
616
例外をスロー
614
Y
N
606
612
保護
オペレーション？
Y
セキュリティマネー
ジャを呼び出す
610
オペレーション
が許されるか？
N
Y
オペレーションを
実行
608

【図９】


610
開始
702
オペレーションを識別
704
リソースの名前を取得
706
構成ファイルから対応するアクセスファイルの名前を取得
708
アクセスファイルからアプレットに対応する許可を取得
710
終了
712

【図10】



フロントページの続き

(72)発明者　スティーヴン　ビー．　バーン
アメリカ合衆国，　カリフォルニア州，
サン　ノゼ，　カーター　アヴェニュー
5269

【外国語明細書】

1. Title of Invention

METHOD AND APPARATUS FOR CONTROLLING SOFTWARE ACCESS TO SYSTEM RESOURCES

2. Claims

IN THE CLAIMS:

1 A method for controlling the degree of access to operating system resources for a software program running on a computer which computer is running said operating system, the method comprising the steps of:

(a) defining said degree of access to said operating system resources for said software program;

(b) examining at least one file associated with said software program to determine the degree of system-level access available to said software program when said software program is being executed by said computer;

(c) executing said software program on said computer;

(d) intercepting a program instruction associated with said software program when said software program is being executed on said computer;

(e) determining if said program instruction includes an operation that is outside said degree of system-level access available to said software program; and

(f) executing said program instruction when it is determined that said software program has permission to access system-level resources associated with said computer that are within the degree of system-level access available to said software program.

2. A method as recited in claim 1 wherein said step of determining if said program instruction includes an operation that is outside said degree of system-level access available to said software program comprises validating an identifier associated with said software program.

3. A method as recited in any one of the preceding claims wherein said step of executing said program instruction comprises determining if said system-level

resources being accessed by said program instruction are protected system-level resources.

4. A method as recited in any one of the preceding claims wherein said software program comprises an applet.

5. A method as recited in claim 4 wherein said applet is a Java applet.

6. A method as recited in one of claims 4 and 5 wherein said applet is associated with a header, said header being arranged to include an identifier, said identifier being arranged to identify said an origin of said file.

7. A method as recited in claim 6 further including the step of validating said identifier to determine if said computer has permission to access said system-level resources.

8. A method as recited in one of claims 4-7 wherein said computer is a client computer and said applet is downloaded to said client computer from a server computer.

9. A method as recited in claim 8 wherein:

(a) said step of examining includes determining the degree of system-level access to said server that is available to said applet when said applet is being executed by said client computer as defined by said defining a degree of access to said system-level resources associated with said server computer for said applet;

(b) said step of determining includes determining if said program instruction to access system-level resources associated with said server computer includes

an operation that is outside said degree of system-level access available to said applet; and

(c) said step of executing includes executing said program instruction to access system-level resources associated with said server computer when it is determined that said applet has permission to access system-level resources associated with said server computer that are within the degree of system-level access available to said applet.

10. A method for controlling the degree of access to operating system resources for a software program running on a client computer which client computer is running said operating system, wherein at least some of said operating system resources reside on a server computer that is coupled with said client computer through a computer network, the method comprising the steps of:

(a) defining said degree of access to said operating system resources for said software program;

(b) loading at least one file including instructions for executing said software program on said client computer;

(c) examining said at least one file to determine the degree of system-level access available to said software program when said software program is being executed by said client computer as defined by said step of defining said degree of access;

(d) executing said software program on said client computer;

(e) intercepting a program instruction associated with said software program when said software program is being executed on said client computer;

(f) determining if said program instruction includes an operation that is outside said degree of system-level access available to said software program; and

(g) executing said program instruction when it is determined that said software program has permission to access system-level resources that are within the degree of system-level access available to said software program.

11. A method as recited in claim 10 wherein said step of determining if said program instruction includes an operation that is outside said degree of system-level access available to said software program comprises validating an identifier associated with said software program.

12. A method as recited in one of claims 10 and 11 wherein said step of executing said program instruction comprises determining if said system-level resources being accessed by said program instruction are protected system-level resources.

13. A method as recited in one of claim 10-12 further including the steps of:

establishing a data transfer communication link between said client computer and said server computer across said computer network; and

transmitting said at least one file from said server computer to said client computer across said computer network.

14. A method for processing a request from a client to access a system resource associated with a first server, the method comprising the steps of:

(a) calling a second server to initiate a download of files that are relevant to said request;

(b) loading said relevant files from said second server, said relevant files including an archive file, said archive file including at least one class file and a header, said header including an identifier arranged to indicate the origin of said archive file;

(c) validating said archive file;

(d) converting said class file into an applet; and

(e) executing said applet, said applet including at least one instruction, wherein executing said applet enables said client to access said system resource associated with said first server.

15. A method for processing a request as recited in claim 14 wherein said step of validating said archive file includes the sub-steps of:

(a) authenticating said header;

(b) determining whether said header is valid; and

(c) performing a class verification on said class when it is determined that said header is valid.

16. A method for processing a request as recited in one of claims 14 and 15 wherein said step of executing said applet includes the sub-steps of:

(a) determining whether said instruction is an instruction to execute a protected operation;

(b) executing said operation when it is determined that said instruction is not an instruction to execute a protected operation; and

(c) determining whether said operation is allowed when it is determined that said instruction is an instruction to execute a protected operation.

17. A computer system for controlling the degree of access to operating system resources comprising:

a first computer coupled with at least one memory device which holds therein at least one file including instructions for executing a software program, said software program running on said first computer, said first computer running said operating system, said first computer being configured to:

(a) define said degree of access to said operating system resources for said software program and said first computer being configured determine if a program instruction associated with said software;

(b) load said at least one file including instructions for executing said software program on said first computer;

(c) examine said at least one file to determine the degree of system-level access available to said software program when said software program is being executed by said first computer;

(d) execute said software program on said first computer;

(e) intercept a program instruction associated with said software program when said software program is being executed on said first computer;

(f) determine if said program instruction includes an operation that is outside said degree of system-level access available to said software program; and

(g) execute said program instruction when it is determined that said software program has permission to access system-level resources associated with said first computer that are within the degree of system-level access available to said software program.

18. A computer system according to claim 17 wherein said first computer is arranged to determine if said system-level resources being accessed by said program instruction are protected system-level resources.

19. A computer-readable medium comprising computer-readable program code devices configured to cause a computer to perform the actions of:

(a) defining said degree of access to said operating system resources for said software program;

(b) examining at least one file associated with said software program to determine the degree of system-level access available to said software program when said software program is being executed by said computer;

(c) executing said software program on said computer;

(d) intercepring a program instruction associated with said software program when said software program is being executed on said computer;

(e) determining if said program instruction includes an operation that is outside said degree of system-level access available to said software program; and

(f) executing said program instruction when it is determined that said software program has permission to access system-level resources associated with said computer that are within the degree of system-level access available to said software program.

整理番号：P97HI－025 (9／30)

3. Detailed Description of Invention

BACKGROUND OF THE INVENTION

1. Field of Invention

The present invention relates generally to methods and apparatus for controlling the access to computer resources by software running on a computer. More specifically, the present invention relates to methods and apparatus for controlling the access to system resources on a client computer by software downloaded to the client computer from a server computer.

2. Background

Prior to the rise of the personal computer, computer users were limited to operating software that ran on large, mainframe computers using terminals that typically included a keyboard for entering data and commands and a video display device (or printer) for viewing output. Although mainframes provided very powerful computing platforms, they suffered from serious drawbacks. In particular, mainframes were expensive to install and operate and they required all users to be connected directly to the mainframe through a terminal, which limited access to the mainframe for many people. In addition, users had very limited control over their computing environments, usually having to adapt their work styles and problems to suit the software and administration of the mainframe computer.

Beginning in the late 1970's personal computers began to overtake mainframes as the dominant computing platform for both personal, business, and scientific uses. For single users, personal computers often could provide the same computing speed as the older mainframes that had to accommodate many processing jobs simultaneously.

In addition, software that ran on the personal computers became more "user-friendly," thereby allowing computer users to adapt both the computer and the software to suit their particular computation needs. The release from requiring a connection from a terminal to a mainframe allowed personal computers to be located just about anywhere within an organization or at home. This capability further assured the dominance of the personal computer over the mainframe as computing power could be located at sites where it was needed. No longer did users have to tailor their operations around large, expensive, finicky mainframe computing centers.

As the computing power and data storage capacities of personal computers exploded throughout the 1980s, the dominance of the personal computer seemed to be assured. As the 1980s drew to a close, however, a new phenomenon began to emerge which appears likely to overtake the personal computer revolution of the past two decades. Today, ever increasing numbers of personal computers are linked to each other through high speed data networks. The most popular network currently is the "Internet," which is the network comprising various business, academic, and personal computer sites across the globe. The popularity of the Internet, and, more particularly, that aspect of the Internet referred to as the "World Wide Web," has prompted many organizations to form internal computer networks, which are often referred to as "intranets." This interest in network computing has been sparked by a combination of high speed data networks and increasingly sophisticated network servers, routers and other devices which allow many independent personal computers to communicate efficiently.

The attractiveness of the World Wide Web stems in part from its highly visual character, the same factor that played a large role in the rise of the personal computer and its dominance over the mainframe. Typically, the World Wide Web is organized into various "web sites" which typically comprise a server that transmits data to a client computer running a "browser." The browser is software that provides a user with a

window and various controls through which data from the server can be viewed and navigated. A particularly useful feature of World Wide Web data is its ability to be linked through hypertext commands such that users can quickly navigate from one document to another and even from one web site to another through very simple intuitive commands such as the activation of a mouse button. Using the World Wide Web, users can view and/or download text, graphics and hear sounds from sites all over the globe. In addition users can also download new software, or software capable of modifying programs already installed on the client computers.

These same features available to users of the World Wide Web on the Internet can also be provided to users of a local network through an "intranet", a non-public computer network that includes clients and servers arranged analogously to the Internet. This capability has received increasing attention from many organizations as information useful to employees carrying out their assignments can be distributed quickly throughout the network to personal computers within the organization. In particular, many organizations are utilizing intranets to provide access to databases and custom software programs for individuals in the organization using such intranets. For example custom software applets created using the Java™ programming language (available commercially from Sun Microsystems of Palo Alto, California), can be operated in conjunction with software and data already installed on the remote computer which is either external or internal to the intranet to provide users access to data and software specific to their job tasks without the difficulties associated with disseminating and maintaining many copies of special-purpose software as has been done traditionally.

It is often desirable for software distributed through a secure intranet to have full access to the system resources of the client computer; whereas software distributed over less secure networks external to the intranet system generally are allowed little or no access to system resources, such as file moving capabilities, as such software

cannot always be trusted. For example, some software applications include functions that install computer viruses on the host computer. Other software applications may copy, alter, or delete critical data from the host computer and even forward that data to another computer system surreptitiously. Unfortunately, there is no viable method or apparatus to enable trusted software to access certain resources while restricting other software from accessing the same resource. Users are therefore left with a trade-off between enabling all software (trusted or suspect) access all system resources or limiting the access of all software in an effort to preserve the security of the client system

Thus, it would be of great benefit to computer users, and especially computer users within organizations in which multiple computer users are connected through a computer network, to provide methods and systems for controlling resource access for both information and software over the network so that the above-described problems associated with highly decentralized computer networks can be mitigated. As will be described here and below, the present invention meets these and other needs.

SUMMARY OF THE INVENTION

The present invention addresses the above-described difficulties in managing software distribution across networked computers by providing, in one aspect, a method, system, and software for controlling the access to server resources by selected software applications on a first computer acting as a client computer that is in communication with a second computer acting as a server computer on a computer network.

In one aspect of the present invention, a method for controlling the degree of access to operating system resources for a software program running on a computer. The degree of access to the operating system resources is defined for the software

program, and at least one file including instructions for executing the software program is loaded on the computer. The file is examined to determine the degree of system-level access available to the software program when the software program is being executed by the computer. The software program is executed, and a program instruction requesting access to secure resources associated with the software program is intercepted when the software is being executed on the computer. A determination is then made to determine if the program instruction includes an operation that is outside of a degree of system-level access that is available to the software program, and if it is determined that the software program has permission to access system-level resources associated with the computer that are within the degree of system-level access available to the software, the program instruction is executed.

In another aspect of the present invention, a method for controlling the degree of access to system resources for a software program running on a client computer that is running the operating system, where at least some of the operating system resources reside on a server computer that is coupled with the client computer, is provided. The degree of access to the operating system resources for the software program is defined, and at least one file including instructions for executing the software program on the client computer is loaded. The file is examined to determine the degree of system-level access available to the software program when the software program is being executed by the client computer. The software program is executed on the client computer, and a program instruction associated with the software program is intercepted when the software program is being executed on the client computer. A determination is made regarding whether the program instruction includes an operation that is outside the degree of system-level access available to the software program, and when it is determined that the software program has permission to access system-level resources that are within the degree of system-level access available to the software program, the program instruction is executed.

These, and other aspects and advantages of the present invention, will become apparent when the Description below is read in conjunction with the accompanying Drawings.

DETAILED DESCRIPTION OF THE DRAWINGS

Certain embodiments of a method and apparatus for controlling the access by applets to system resources will be described below making reference to the accompanying drawings.

An illustration of one network in accordance with the present invention is provided in Fig. 1a. Included in the network illustrated in Fig. 1a are intranets 102 and 104 and an individual external computer shown at 106. The structure of intranets 102 and 104 is described in greater detail below with respect to Fig. 1b. Both the intranets and the user are connected to the computer network through a variety of computer gateways ("G/W"). In some embodiments, the computer network includes the Internet. Referring to Fig. 1a more specifically, intranet 102 is coupled with intranet 104 and user 106 through the Internet which is shown generally at 108. The connection between intranet 102 and the Internet 108 is provided first through a gateway 110 which is coupled with intranet 102 and a "backbone," or high capacity dataline 112. Data from a high capacity line 112 is routed through gateway 114 through the Internet 108 which data passes through a second gateway 116 and into high capacity dataline shown at 118. As will be appreciated by those of skill in the computer network arts, dataline 118 can be the same as dataline 112, or may represent a separate backbone to which various other individuals, or users, and networks are coupled.

Data that travels from intranet 102 through the Internet 108 and over high speed dataline 118 passes through gateway 120 to intranet 104 or through gateway 124 to user 106. Thus, according to the illustrated embodiment, data can be passed among user 106, intranet 104, and intranet 102. In particular, the data may travel through the Internet 108 as just described, or may pass across backbone 118 between user 106 and intranet 104. In some embodiments, intranet 104 and intranet 102 can be coupled directly through network configurations known to those of skill in the art as

"extranets". Extranets are network arrangements in which a given network or individual is coupled with a remote network through a dedicated data connection. This connection may include data that is routed through the Internet, as illustrated in Fig. 1a, or may be a direct data feed, such as through an ISDN or T-1 dataline. Various configurations in addition to methods and materials for establishing such configurations will be apparent to those of skill in the computer network and telecommunications arts.

One embodiment of an intranet, such as illustrated in Fig. 1a at 102 or 104, is provided in Fig. 1b at 50. A typical intranet 50 includes a server 60 which is coupled with clients 62 and 64. In addition, server 60 can be coupled to other client computers such as shown at 70, 72, and 74 through a router, hub or similar data transfer device such as shown at node 68. In addition, remote clients (not shown) can be connected to server 60 either through a direct line or through the use of telephone lines using, *e.g.*, a modem or similar device. In some cases, access to intranet 50 will be controlled to a high degree by a "firewall" configuration which is illustrated by the box 75. The establishment of communications from users external to the firewall, such as remote client 78, can be achieved by traversing a gateway which allows access to the protected server. Such a gateway is illustrated at 76.

Typically, a server provides data and software that is accessible to the various clients which are in communication with the server, either directly or through a device such as a router. The construction, maintenance, and operation of the server, router, and various client machines will be well known to those of skill in the art. In some particular embodiments, server 60 will be configured to provide data that is compatible with browser software such as that used to view data on the World Wide Web. Specifically, the data provided by server 60 will be in the form of pages of data that can be examined using typical browser software. In one embodiment, the server and clients are configured to exchange not only data but computer software in the form of "applets," such as those written in the Java™ programming language available from

Sun Microsystems of Palo Alto, California. "Applets" as used herein are software programs that are configured to be passed from a source computer, typically a server, to a client machine and run in conjunction with software already installed on the client. In one embodiment, the software with which the applet runs is the above-described browser software. Typically, applets provide additional functionalities to browser software by performing various computational tasks which the browser software itself is not configured to perform. Thus, users who download applets can provide the browser software with additional functionalities that are not otherwise available to the browser software. Such additional capabilities can include, *e.g.*, custom interfaces to a database.

In general, a client, as for example client 62, calls into server 60 using a user agent, or a browser. User agents include, but are not limited to, HotJava™, available from Sun Microsystems, Incorporated of Palo Alto, California, and Netscape, available from Netscape Communications Corporation of Mountain View, California. In one embodiment, the user agents generally includes a processing engine which executes the applet code, and a security manager used in the determination of whether an applet has access to certain system resources. Examples of such a security manager are provided herein below.

According to one embodiment, a server, located either on the Internet or within an intranet, provides class libraries which contain class files that define an applet. One example of such a class library is a Java™ class library. Specifically, a server can contain the class files that make up the applet, and the particular Web pages including HTML code that references the applet.

According to one embodiment of the present invention, applets are instantiated from class files that are downloaded from a source computer, or a server, to a client machine. The class files may be grouped together into an archive file. Further, an

archive file can be digitally signed, or otherwise marked, such that the origin of an applet created from the archive file can be reliably determined. The signature of an archive file can then be verified in order to determine which system resources are accessible to the machine on which the applet is executing. The use of signatures enables the access to system resources of the client machine by the applet to be controlled, *e.g.*, by reference to the security status of the server from where the applet originated. By way of example, an applet executing on one client may have different access privileges than the same applet executing on a second client by virtue of the fact that the permissions associated with the applet on each client may be different. This resource access control therefore enables applets associated with secure machines, *e.g.*, machines in the same intranet as the machine which contains the resources, to have more access to resources than applets associated with unsecure machines, *e.g.*, machines on the Internet.

Fig. 2a is a diagrammatic representation of a collection of class files in accordance with an embodiment of the present invention. The format of the collection of class data files, which is generally used on a server, is not arranged to accept signatures. That is, each class file typically defines a class residing on a server. The format is such that the collection includes any number of classes, as for example class "1" 202, class "2" 204, and class "N" 206. A class may be defined as a software construct that defines data and methods, or sequences of statements that operate on the data, which are specific to any applets that are subsequently constructed from that class. In other words, as previously stated, an applet may be constructed by instantiating a previously defined class. It should be appreciated that a single class may be used to construct many applets.

The execution of an applet usually entails requests, or commands, to access system resources. While an applet may contain instructions to access many different system resources, due to security concerns, an applet is either allowed access to all of

the specified system resources or access to none of the specified system resources under present design restraints. As discussed above, this "all-or-nothing" approach to system resource access is often undesirable in that an applet running within an intranet system, for example, is "trusted," *e.g.*, of known origin, while an equivalent applet running externally to the intranet system is considered to be unsecure. As the applet running within the intranet system and the equivalent applet running externally are typically given the same access privileges to system resources, in order to maintain the security of the intranet system, the applets are generally given no access privileges.

The ability to selectively control applets from accessing resources enables a user within an intranet system to restrict access to resources on an individual applet basis. Including a "signature," or an identifier, with class files that are used to instantiate an applet is one method which serves to enable an intranet organization to selectively control applets. Signing, or marking, class files such that it is possible to determine where the class files originated enables an intranet system to determine the appropriate access privileges associated with an applet instantiated from the class files. In addition, signing class files further enables a determination to be made regarding whether a class file has been tampered with. An archive file structure which permits a group of class files to be digitally signed will be described below with respect to Fig. 2b.

By providing an archive file which can be digitally signed, it becomes possible to enable an applet, either internal and external to an intranet system, that is constructed from the class files associated with the archive file to access selected system resources within the intranet system. Checking the digital signature of the archive file makes it possible to determine whether a given applet has been tampered with, and which computers have signed the applet. As such, access privileges may be allocated based upon whether the applet originated from a secure, or trusted, host or from an unsecure host. In addition, in some embodiments, the allocation of access privileges enables users to decide which hosts are to be trusted and which are not to be trusted.

Fig. 2b is a diagrammatic representation of an archive file data format in accordance with an embodiment of the present invention. In the described embodiment, the archive format is a Java™ archive (JAR) format. Archive, or archive file, 210 includes a header signature 212 which is the signature that is typically used by a user agent to verify the validity of archive 210 and to determine the levels of access available to archive 210. In general, header signature 212 is a digital signature which may be a part of a general header that contains other information which information includes, but is not limited to, information corresponding to the size of the archive. Archive 210 has any number of associated classes, as for example class "1" 202, class "2" 204, and class "N" 206, from which applets and associated objects are instantiated.

Additionally, archive 210 may have associated data blocks, as for example data block 214. Data block 214 may contain images, text, or any arbitrary data that is considered to be a part of archive 210. In one embodiment, data block 214 may contain a text string that describes classes 202, 204, and 206 that are associated with archive 210. It should be appreciated that in other embodiments, archive 210 may not include a data block.

Referring next to Fig. 3a, an embodiment of a client-side directory structure will be described in accordance with the present invention. A user who makes a request to access a resource through a client generally interfaces with a user directory 302. User directory 302 has an associated browser directory 304 which contains information relating to a browser, or a user agent. The browser may be any suitable browser, as for example the HotJava™ browser as mentioned above. Browser directory 304 includes a properties file 306 that is appropriate to the request made by the user. Properties file 306 typically includes user preference items 308 which are generally browser specifications that are provided by the user. These specifications may include,

but are not limited to, data relating to browser set-up and behavioral properties associated with the browser.

Properties file 306 further includes information that is relevant to the particular request made by the user. By way of example, such information can include an images data block 310, a configuration file name 312, and a group specification file name 314. In one embodiment, images data block 310 includes data file names, *i.e.*, strings, which identify any images that are associated with the request. A configuration file name 312 is a string that identifies a configuration file which is used to facilitate the mapping of a requested resource to associated security descriptors. One example of a configuration file will be described below with reference to Fig. 3b. Group specification ("spec") file name 314 is a string which identifies a group specification file, as will be described below with respect to Fig. 3c.

Fig. 3b is a diagrammatic representation of the structure of a configuration file in accordance with an embodiment of the present invention. Configuration file 350 is an example of a configuration file identified by configuration file name 312 as mentioned above with respect to Fig. 3a. Configuration file 350 includes a table 352 which associates resources 354 on a server, *i.e.*, a server which the client wishes to access, with corresponding access file names 356. That is, table 352 associates an entry in the resources "column" 354 with a corresponding entry in the access file names "column" 356. Resources 354 are generally classifiers which identify various system resources, as for example files, hosts, and socket numbers. Access file names 356 identify corresponding access files which contain security descriptors and other information that is relevant to the control of access to system resources with which access files are associated. The structure of an access file will be described in more detail below with reference to Fig. 3c. It should be appreciated that due to the fact that more than one resource 354 may share the same security descriptor, access file names 356 and, therefore, access files, may be associated with more than one resource 354.

Referring next to Fig. 3c, the structure of an access file will be described in accordance with an embodiment of the present invention. Access file 360 generally includes a table 361 which associates principals 362 with permissions 364. Principals 362 may be individual hosts or groups of hosts. By way of example, "java.com" may be an individual host, *i.e.*, a server, which is a principal 362. Alternatively, "java.com" and "sun.com" may form a principal 362 that is a group. In some embodiments, principals 362 can also be the signers of particular archives. Permissions 364 provide groupings of security descriptors. That is, permissions 364 are groupings of security descriptors which designate the resources that principals 362, with which permissions 364 are associated, have access.

Fig. 3d is a diagrammatic representation of a group specification ("spec") file format in accordance with an embodiment of the present invention. As mentioned above, the group specification file name 314 of Fig. 3a identifies a group specification file, as for example group specification file 370. Group specification file 370 includes a table 371 that associates group names 372 with any number of members 374. Group names 372 are essentially identifiers that may be used to identify a group of member 374. By way of example, a group name, as for example group "1" 372a, may be associated with any number of members, as for example member "1" 374a and member "2" 374b. It should be appreciated that a member, as for example member "1" 374a, may be associated with more than one group name 372.

Fig. 4 is a process flow diagram which illustrates a method of executing a request to access a resource in accordance with an embodiment of the present invention. The process begins at 402 and in a step 404, a call is made from a requesting client, *e.g.*, client 74 of Fig. 1b, to a server, *e.g.*, server 60 of Fig. 1b, to initiate the download of either at least one class file, as described above with respect to Fig. 2a, or an archive file, as described above with respect to Fig. 2b. The request is received on

the server in response to a client call made through a user agent, *i.e.*, a browser, as for example a HotJava™ browser or a Netscape Navigator browser as previously mentioned. The initiation of the downloading of either at least one class file or an archive file occurs in response to a request to access a resource and, hence, is a call to execute an applet. In one preferred embodiment, the archive file is a JAR file.

In a step 406, either the archive file is loaded or the class files are loaded from the server into memory associated with the requesting client. In general, class files are loaded if the classes are not in an archive file, *e.g.*, not digitally signed, and an archive file is loaded if the classes are digitally signed. It should be appreciated that the archive file has associated class files. As such, loading the archive file involves loading class files. After the class files are loaded into memory, a validation process is performed on the loaded files in a step 408. The validation process, which includes the process of verifying whether the header signature associated with a loaded archive file is valid, in the event that an archive file has been loaded, will be described below with reference to Fig. 5.

After the validation process, in a step 410, the class files are converted into an applet. That is, an applet is created in memory by instantiating the loaded class files, which may or may not be a part of a JAR file. Once the applet is created, the applet file is executed in a step 412. The steps associated with the execution of an applet will be described below with respect to Fig. 6.

Fig. 5 is a process flow diagram which illustrates the steps associated with validating class files, *i.e.*, step 408 of Fig. 4, in accordance with an embodiment of the present invention. The process begins at step 502, and in a step 504, a determination is made regarding whether an archive file or a class file has been loaded. If a class file has been loaded, then process flow proceeds to a step 506 in which a standard class verification is performed. A standard class verification typically includes a check of all

loaded class files and, therefore, classes, in order to ascertain whether anything in the class files may compromise security. In some embodiments, a check is made to determine if the security of a virtual machine, as for example a Java™ virtual machine, can be compromised. Standard class verification methods are generally well known to those of ordinary skill in the art. Once the standard class verification is performed, the process of validating the class files is completed at 520.

If the determination in step 504 is that an archive file has been loaded, then in a step 508, the header of the archive file is validated, or authenticated. The validation of the archive file generally involves an identification of the origin of the archive file based upon the header signature. That is, a check is made to establish the origin of the header signature and, therefore, the archive file. The validation may also include a check of whether data associated with the archive file is intact. It should be appreciated that in some embodiments, an archive file may not include a header signature. By way of example, an archive file within an intranet may not be signed. In a step 510, a determination is made as to whether the header is valid. If the header is not valid, *e.g.*, the content of the archive does not correspond with the signature, then in a step 514, an error flag or the like is raised. In one embodiment, the error flag may result in an exception being thrown. In another embodiment, the error flag may result in a message being returned to the requesting client. After the error flag is raised, the process of validating class files ends at 520.

If the header is found to be valid in step 510, process flow moves from step 510 to a step 512 which is the determination of whether any classes associated with the archive file remain to be validated. If there is a class to be validated, then in a step 516, a standard class verification is performed. As previously described in step 506, a standard class verification includes a check of whether anything in a given class may compromise the security of a virtual machine. By way of example, the security of a virtual machine may be compromised if something in a given class can overwrite files

or memory on the virtual machine. After the standard class verification is completed on the given class, process control returns to step 512 in which a determination is made regarding whether there are any more classes which are to be validated. Process control loops between steps 512 and 516 until a determination is made in step 512 that no more classes remain to be validated, at which point the process of validating class files is completed at 520.

Fig. 6 is a process flow diagram which illustrates the steps associated with executing an applet in accordance with an embodiment of the present invention. That is, step 412 of Fig. 4 will be described. The process begins at 602, and, in a step 604, a determination is made as to whether the applet contains an instruction to execute an operation. The operation may generally be a call to access a system-level resource. If the applet does not contain an instruction to execute an operation, then the process of executing the applet ends at 616. If the applet does contain an instruction to execute an operation, then process flow proceeds to a step 606 in which it is determined whether the operation to be executed is a protected, *e.g.*, secured, operation. That is, a determination is made regarding whether the operation is an operation to which access is controlled. If it is determined that the operation is not protected, then the operation is executed in a step 608, and process flow returns to step 604, which is the determination of whether there is an instruction to execute another operation.

If it is determined in step 606 that the operation in the instruction to execute is protected, then process flow moves to a step 610 in which the applet security manager is called. The process of calling the security manager will be described in more detail below with reference to Fig. 7. The applet security manager typically controls the operations which are accessible to given applets. In one embodiment, the applet security manager is a Java™ applet security manager. In a step 612, it is determined whether the operation is allowed. In other words, step 612 is the determination of whether the applet has access to the operation which is to be executed. If the operation

is allowed, then the operation is executed in step 608. From step 608, process control returns to step 604 which is the determination of whether there is an instruction to execute another operation.

If the determination in step 612 is that the operation is not allowed, then an error condition occurs, which can be implemented by having an exception is thrown in step 614, and the process of executing the applet ends at 616. It should be appreciated that in some embodiments, the step of throwing an exception may involve calling a throw function. In other embodiments, the step of throwing an exception may involve transmitting an error message which may be displayed by a user agent that is associated with the requesting client. In still other embodiments, the error handling may cause an interaction with the user to occur in the form of asking whether the user approves the performance of the operation by the applet. In such embodiments, access files can possibly be updated to permanently record the response provided by the user.

Referring next to Fig. 7, the process of calling a security manager, *i.e.*, step 610 of Fig. 6, will be described. It should be appreciated that a user agent generally has only one associated security manager. The process of calling a security manager begins at 702 and in a step 704, the operation which is being called by the applet is identified. Although the operation may be any one of a number of operations, the operation is generally a read operation or a write operation. From step 704, process flow proceeds to a step 706 in which the name of the resource associated with the operation is identified. In some embodiments, the name of the resource is passed into the call to the security manager and, hence, is readily identified. However, when the name of the resource is not passed into the call, the properties file, as previously described with respect to Fig. 3a, may be used to identify the associated resource.

Once the associated resource is identified in step 706, the name of the access file which corresponds to the resource is identified using the configuration file, which was

described earlier with respect to Fig. 3b, that is associated with the applet. Permissions corresponding to the applet are then obtained from the access file in a step 710. It should be appreciated that in some embodiments, the appropriate access file may be a representation of the actual access file in memory. The access file, as described above with respect to Fig. 3c, associates individual hosts or groups with a set of permissions. After the permissions are obtained, the call to the security manager is completed at 712.

Fig. 8 illustrates a typical computer system in accordance with the present invention. The computer system 830 includes any number of processors 832 (also referred to as central processing units, or CPUs) that is coupled to memory devices including primary storage devices 834 (typically a read only memory, or ROM) and primary storage devices 836 (typically a random access memory, or RAM). As is well known in the art, ROM 834 acts to transfer data and instructions uni-directionally to the CPU and RAM 836 is used typically to transfer data and instructions in a bi-directional manner. Both primary storage devices 834, 836 may include any suitable computer-readable media as described above. A mass memory device 838 is also coupled bi-directionally to CPU 832 and provides additional data storage capacity. The mass memory device 838 may be used to store programs, data and the like and is typically a secondary storage medium such as a hard disk that is slower than primary storage devices 834, 836. Mass memory storage device 838 may take the form of a magnetic or paper tape reader or some other well-known device. It will be appreciated that the information retained within the mass memory device 838, may, in appropriate cases, be incorporated in standard fashion as part of RAM 836 as virtual memory. A specific mass storage device such as a CD-ROM 834 may also pass data uni-directionally to the CPU.

CPU 832 is also coupled to one or more input/output devices 840 that may include, but are not limited to, devices such as video monitors, track balls, mice, keyboards, microphones, touch-sensitive displays, transducer card readers, magnetic or paper tape readers, tablets, styluses, voice or handwriting recognizers, or other well-

known input devices such as, of course, other computers. Finally, CPU 832 optionally may be coupled to a computer or telecommunications network, *e.g.*, an Internet network or an intranet network, using a network connection as shown generally at 812. With such a network connection, it is contemplated that the CPU might receive information from the network, or might output information to the network in the course of performing the above-described method steps. The above-described devices and materials will be familiar to those of skill in the computer hardware and software arts. Further, it should be appreciated by those skilled in the art that the above described hardware and software elements, as well as networking devices, are of standard design and construction.

The computer-implemented methods described herein can be implemented using techniques and apparatus that are well-known in the computer science arts for executing computer program instructions on computer systems. As used herein, the term "computer system" is defined to include a processing device (such as a central processing unit, CPU) for processing data and instructions that is coupled with one or more data storage devices for exchanging data and instructions with the processing unit, including, but not limited to, RAM, ROM, CD-ROM, hard disks, and the like. The data storage devices can be dedicated, *i.e.*, coupled directly with the processing unit, or remote, *i.e.*, coupled with the processing unit over a computer network. It should be appreciated that remote data storage devices coupled to a processing unit over a computer network can be capable of sending program instructions to a processing unit for execution on a particular workstation. In addition, the processing device can be coupled with one or more additional processing devices, either through the same physical structure (*e.g.*, a parallel processor), or over a computer network (*e.g.*, a distributed processor.). The use of such remotely coupled data storage devices and processors will be familiar to those of skill in the computer science arts (*see*, *e.g.*, Ralston 1993). The term "computer network" as used herein is defined to include a set of communications channels interconnecting a set of computer systems that can

communicate with each other. The communications channels can include transmission media such as, but not limited to, twisted pair wires, coaxial cable, optical fibers, satellite links, or digital microwave radio. The computer systems can be distributed over large, or "wide," areas (*e.g.*, over tens, hundreds, or thousands of miles, WAN), or local area networks (*e.g.*, over several feet to hundreds of feet, LAN). Furthermore, various local-area and wide-area networks can be combined to form aggregate networks of computer systems. One example of such a confederation of computer networks is the "Internet".

Although only a few embodiments of the present invention have been described, it should be understood that the present invention may be embodied in many other specific forms without departing from the spirit or the scope of the present invention. By way of example, although only one configuration of an archive file data structure which may be signed has been described, it should be appreciated that the archive file data structure may be widely varied within the scope of the present invention. Further, steps involved with a method of executing a request to access system resources may be reordered. Steps may also be removed or added without departing from the spirit or the scope of the present invention. Therefore the described embodiments should be taken as illustrative and not restrictive, and the invention should be defined by the following claims and their full scope of equivalents.

4. Brief Description of Drawings

The invention, together with further advantages thereof, may best be understood by reference to the following description taken in conjunction with the accompanying drawings in which:

Fig. 1a is a diagrammatic representation of a wide area computer network in which both users and intranets are coupled by a computer network through the Internet.

Fig. 1b is a diagrammatic representation of a conventional intranet system.

Fig. 2a is a diagrammatic representation of a collection of class files in accordance with an embodiment of the present invention.

Fig. 2b is a diagrammatic representation of an archive file data format in accordance with an embodiment of the present invention.

Fig. 3a is a diagrammatic representation of a client-side directory structure in accordance with an embodiment of the present invention.

Fig. 3b is a diagrammatic representation of the structure of a client-side configuration file in accordance with an embodiment of the present invention.

Fig. 3c is a diagrammatic representation of the structure of a client-side access file in accordance with an embodiment of the present invention.

Fig. 3d is a diagrammatic representation of the structure of a client-side group specification file in accordance with an embodiment of the present invention.

Fig. 4 is a process flow diagram which illustrates a method of executing a request to access a resource in accordance with an embodiment of the present invention.

Fig. 5 is a process flow diagram which illustrates the steps associated with validating class files in accordance with an embodiment of the present invention.

Fig. 6 is a process flow diagram which illustrates the steps associated with executing an applet in accordance with an embodiment of the present invention.

Fig. 7 is a process flow diagram which illustrates the steps associated with calling a security manager in accordance with an embodiment of the present invention.

Fig. 8 is a diagrammatic representation of a computer system in accordance with the present invention.

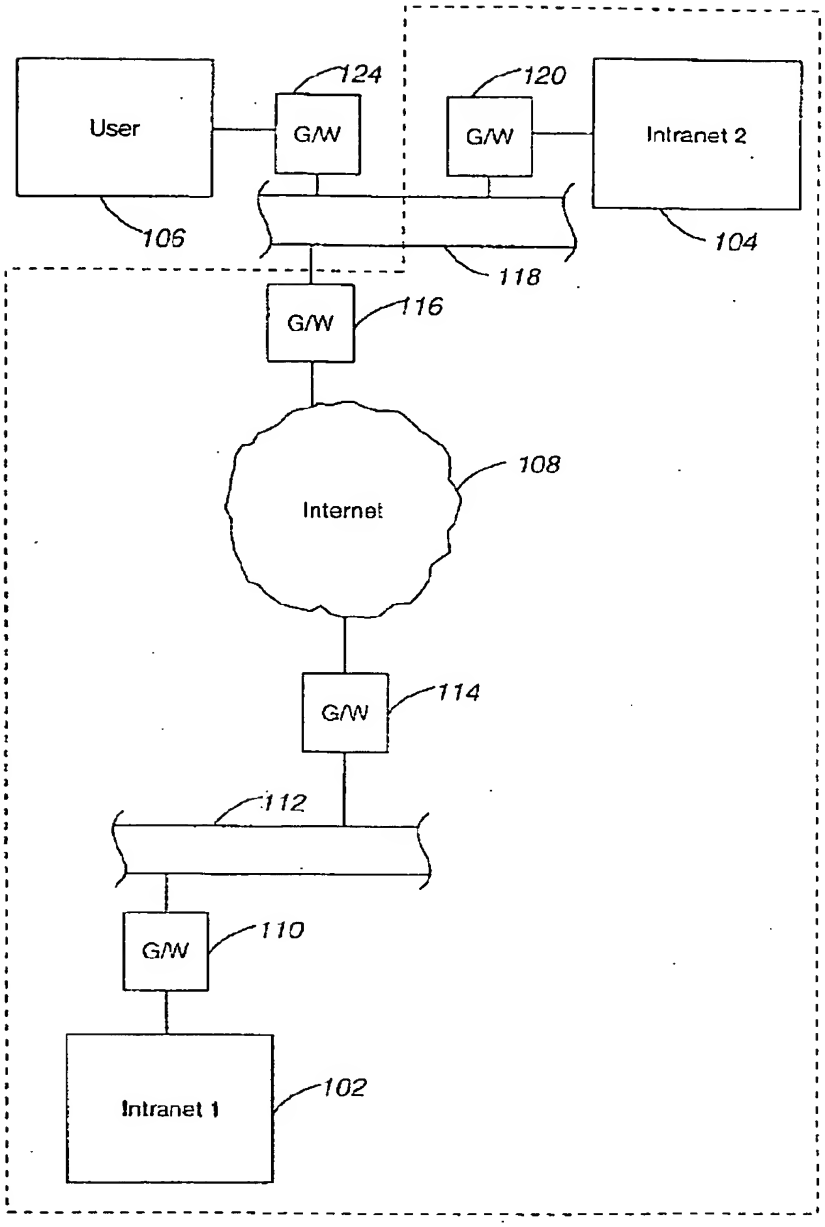

User
G/W
124
G/W
120
Intranet 2
105
104
118
G/W
116
Internet
108
G/W
114
112
G/W
110
Intranet 1
102

Fig. 1a

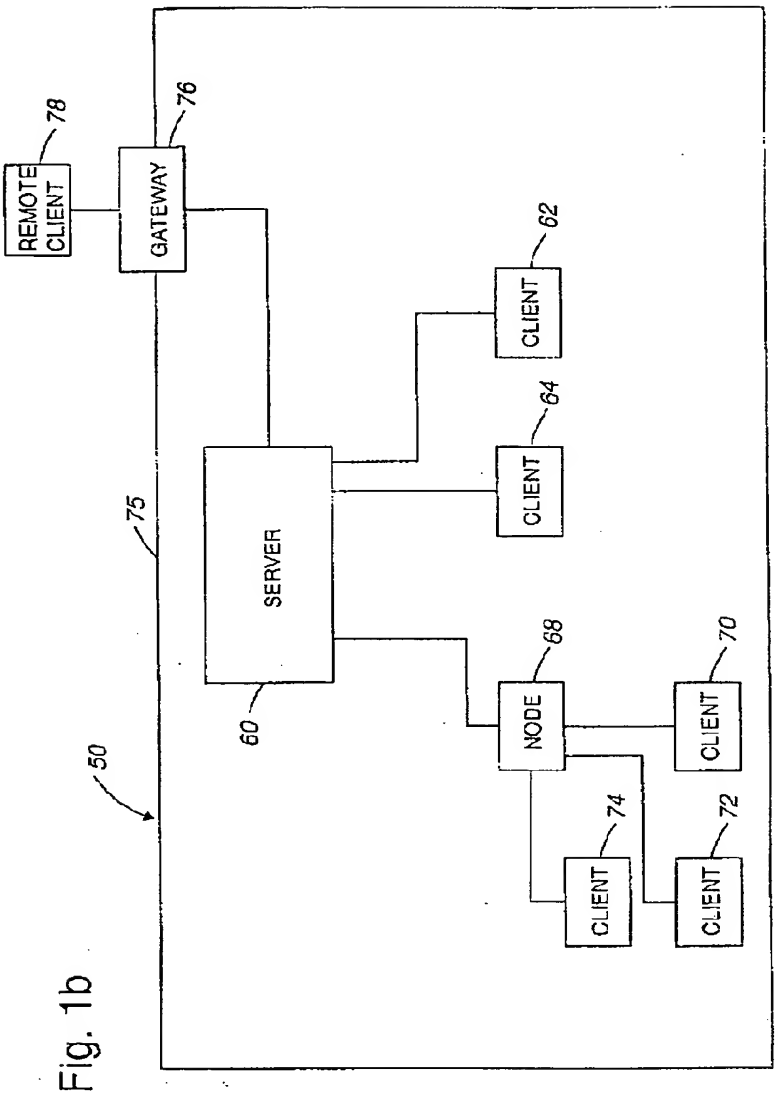

REMOTE CLIENT
78
GATEWAY
76
CLIENT
62
CLIENT
64
SERVER
75
60
NODE
68
CLIENT
70
CLIENT
74
CLIENT
72
50

Fig. 1b

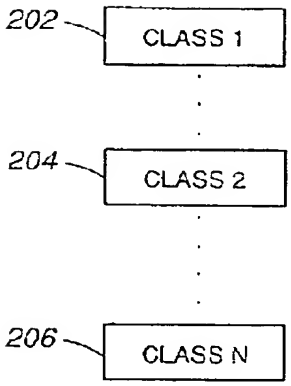


Fig. 2a

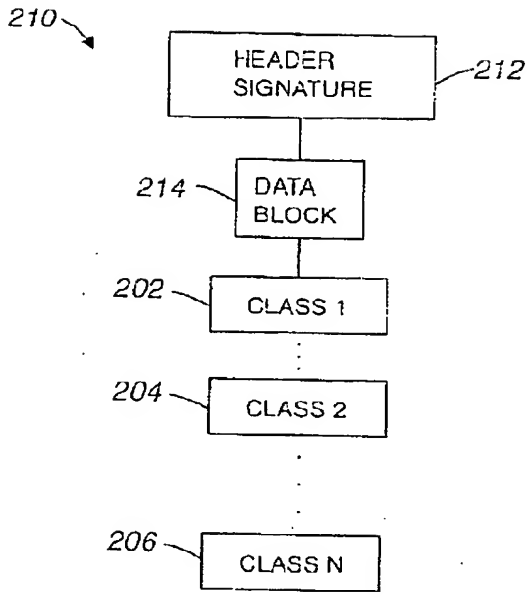

210
HEADER SIGNATURE
212
214
DATA BLOCK
202
CLASS 1
204
CLASS 2
206
CLASS N

Fig. 2b

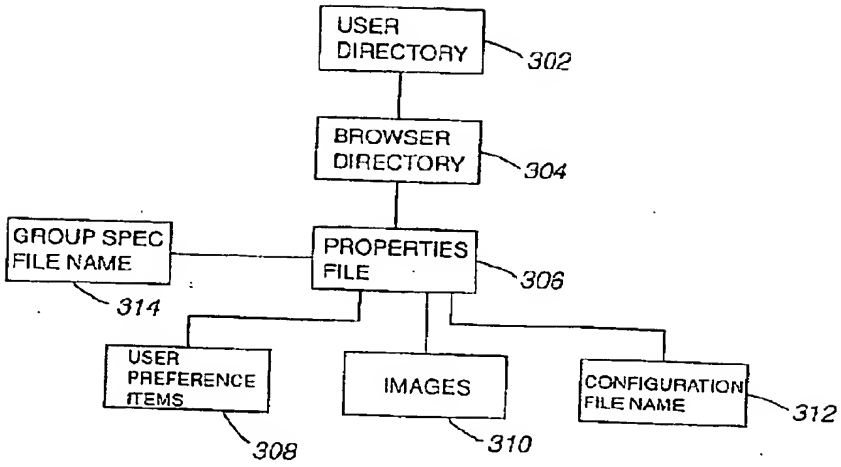

USER DIRECTORY
302
BROWSER DIRECTORY
304
GROUP SPEC FILE NAME
314
PROPERTIES FILE
306
USER PREFERENCE ITEMS
308
IMAGES
310
CONFIGURATION FILE NAME
312

Fig. 3a

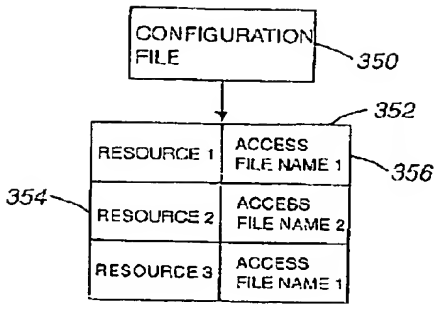

CONFIGURATION FILE
350
352
RESOURCE 1
ACCESS FILE NAME 1
356
354
RESOURCE 2
ACCESS FILE NAME 2
RESOURCE 3
ACCESS FILE NAME 1

Fig. 3b

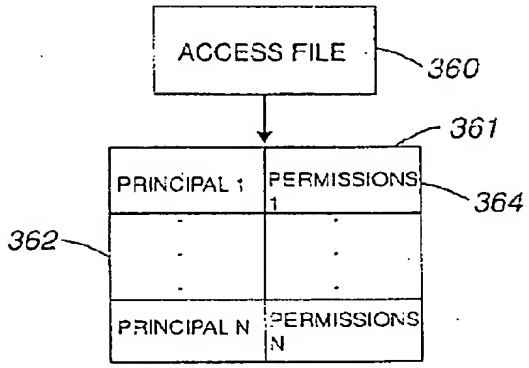

ACCESS FILE
360
361
PRINCIPAL 1
PERMISSIONS 1
364
362
PRINCIPAL N
PERMISSIONS N

Fig. 3c

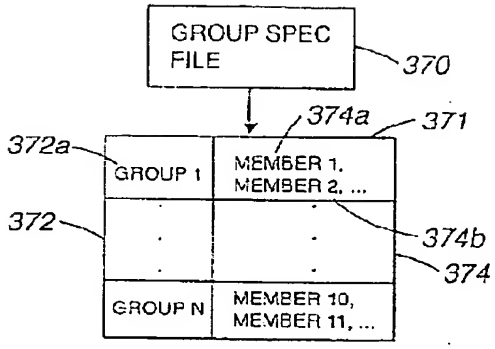

GROUP SPEC FILE
370
374a
371
372a
GROUP 1
MEMBER 1, MEMBER 2, ...
372
374b
374
GROUP N
MEMBER 10, MEMBER 11, ...

Fig. 3d

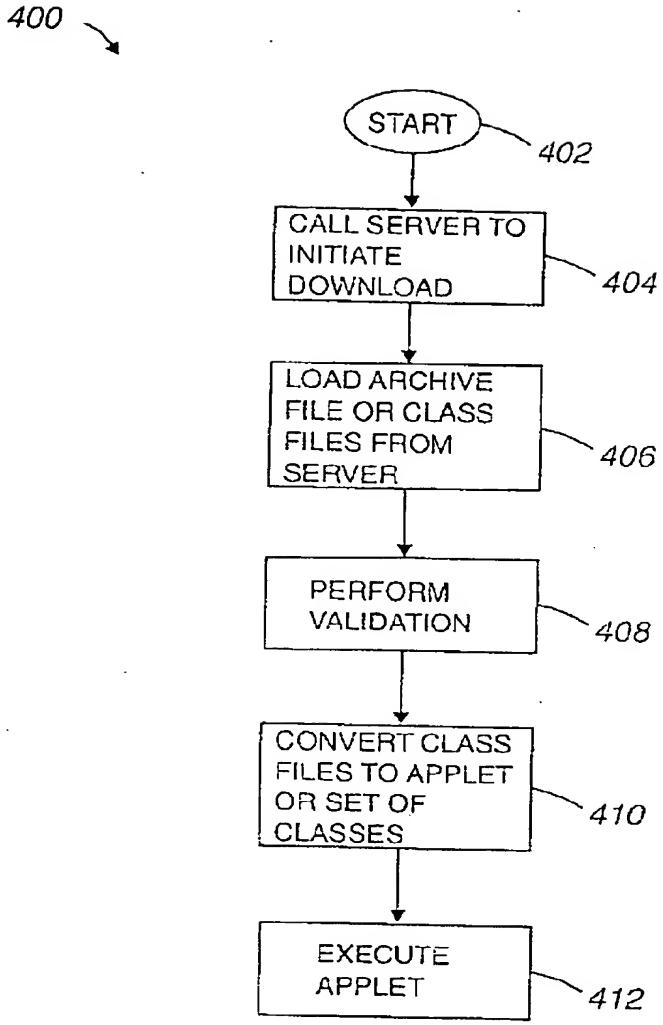

400
START
402
CALL SERVER TO INITIATE DOWNLOAD
404
LOAD ARCHIVE FILE OR CLASS FILES FROM SERVER
406
PERFORM VALIDATION
408
CONVERT CLASS FILES TO APPLET OR SET OF CLASSES
410
EXECUTE APPLET
412

Fig. 4

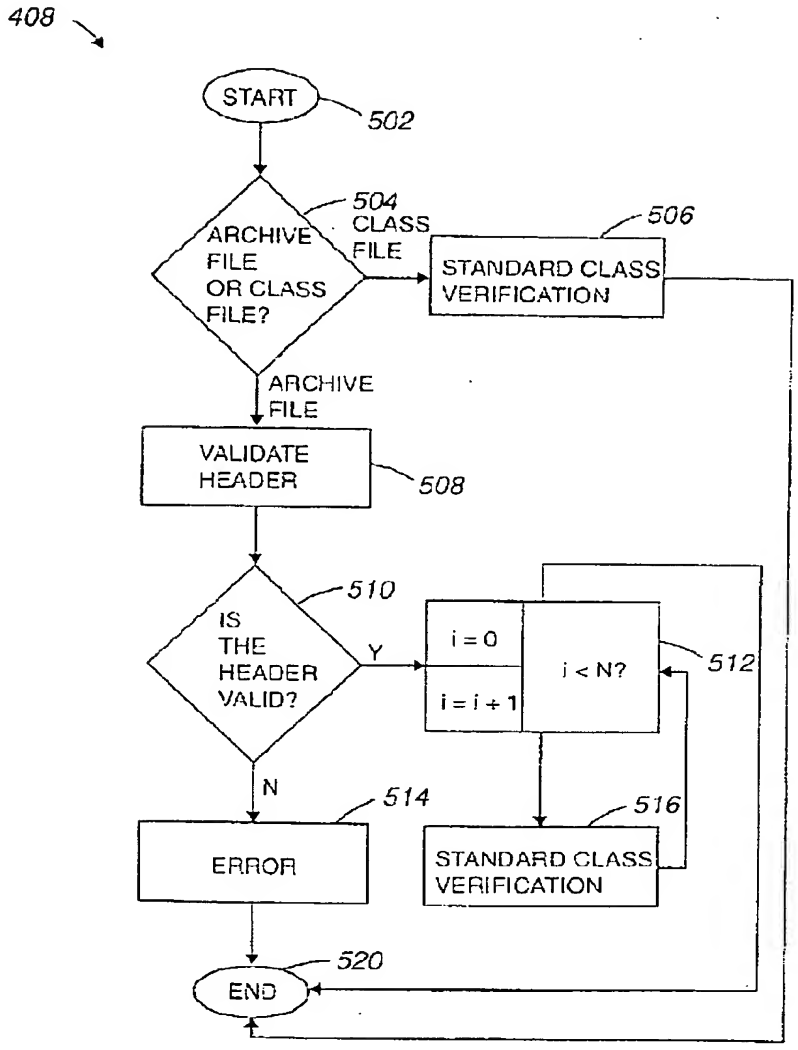

408
START
502
ARCHIVE FILE OR CLASS FILE?
504
CLASS FILE
506
STANDARD CLASS VERIFICATION
ARCHIVE FILE
VALIDATE HEADER
508
IS THE HEADER VALID?
510
Y
i = 0
i = i + 1
i < N?
512
N
514
516
ERROR
STANDARD CLASS VERIFICATION
520
END

Fig. 5

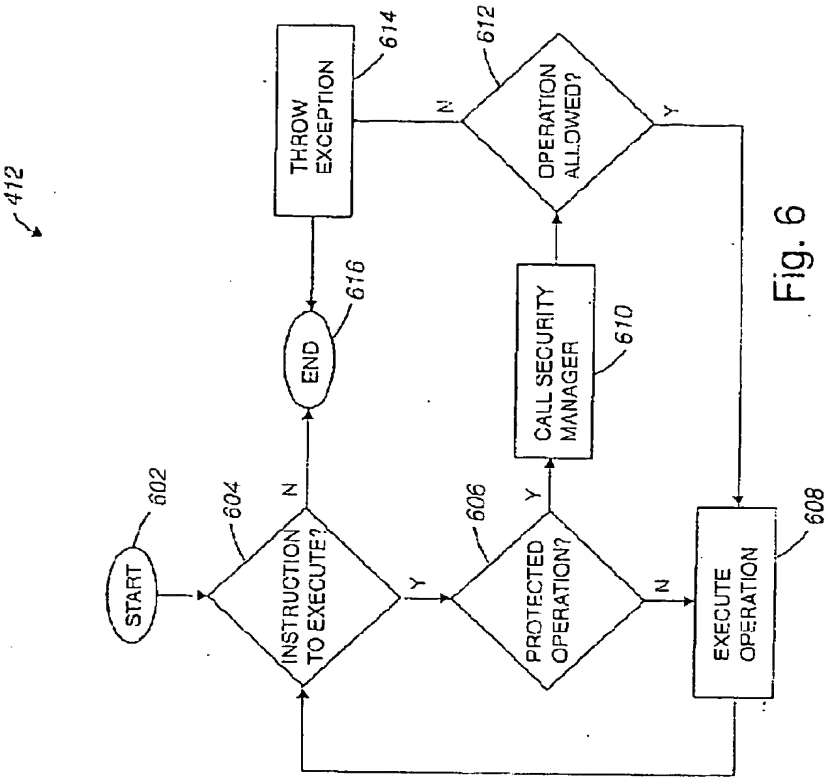

412
START
602
INSTRUCTION TO EXECUTE?
604
N
Y
PROTECTED OPERATION?
606
Y
N
CALL SECURITY MANAGER
610
OPERATION ALLOWED?
612
N
Y
THROW EXCEPTION
614
END
616
EXECUTE OPERATION
608

Fig. 6

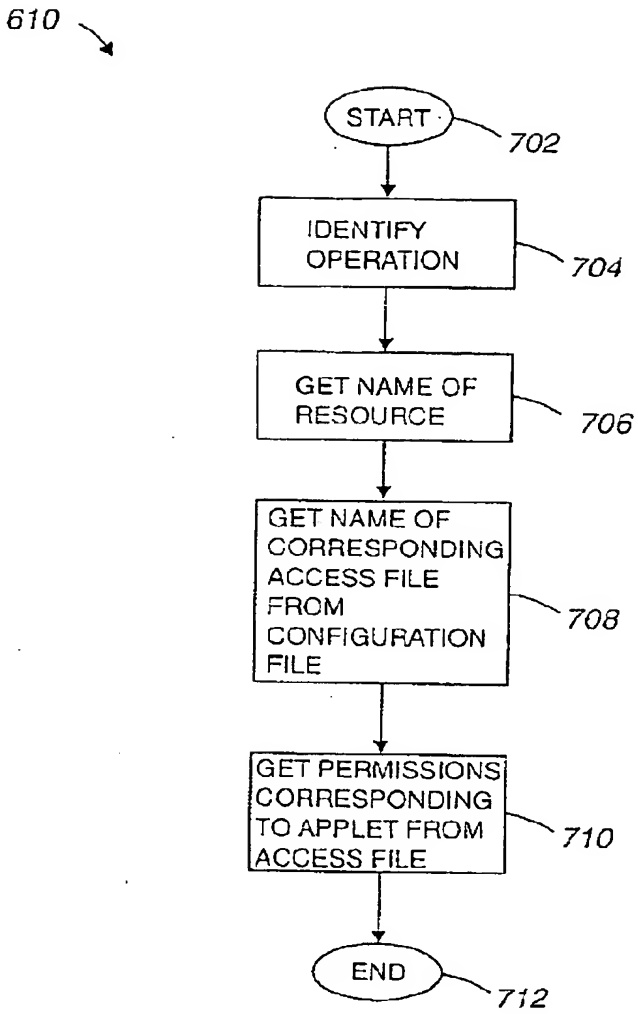

610
START
702
IDENTIFY OPERATION
704
GET NAME OF RESOURCE
706
GET NAME OF CORRESPONDING ACCESS FILE FROM CONFIGURATION FILE
708
GET PERMISSIONS CORRESPONDING TO APPLET FROM ACCESS FILE
710
END
712

Fig. 7

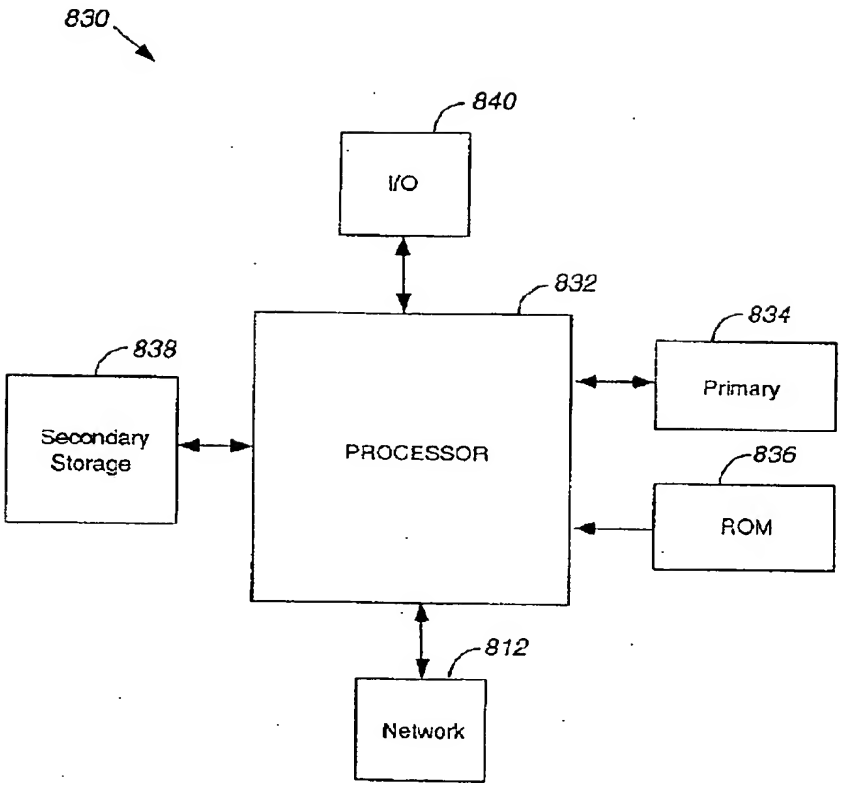

830
840
I/O
832
834
838
Secondary Storage
PROCESSOR
Primary
836
ROM
812
Network

Fig. 8